US 20250265308A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2025/0265308 A1

STANLEY-MARBELL et al. (43) Pub. Date: Aug. 21, 2025

(54) **IMPROVEMENTS IN AND RELATING TO ENCODING AND COMPUTATION ON DISTRIBUTIONS OF DATA**

(71) Applicant: **CAMBRIDGE ENTERPRISE LIMITED**, Cambridge (GB)

(72) Inventors: **Phillip STANLEY-MARBELL**, Cambridge Cambridgeshire (GB); **Vasileios TSOUTSOURAS**, Cambridge Cambridgeshire (GB); **Bilgesu BILGIN**, Cambridge Cambridgeshire (GB)

(73) Assignee: **CAMBRIDGE ENTERPRISE LIMITED**, Cambridge (GB)

(21) Appl. No.: **18/562,405**

(22) PCT Filed: **May 27, 2022**

(86) PCT No.: **PCT/EP2022/064486**

§ 371 (c)(1),
(2) Date: **Nov. 20, 2023**

(30) **Foreign Application Priority Data**

May 27, 2021 (GB) 2107604.7
May 27, 2021 (GB) 2107606.2

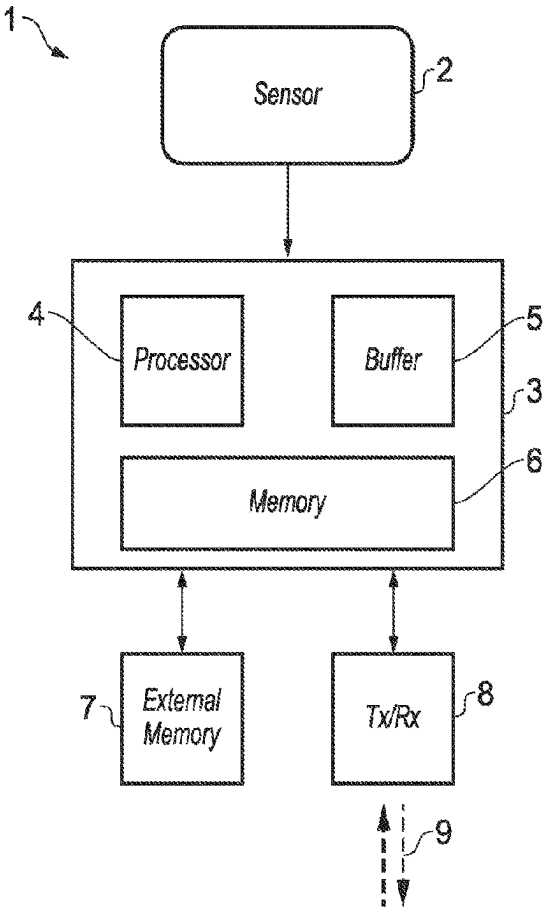
Publication Classification

(51) **Int. Cl.**
G06F 17/18 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/18*** (2013.01)

(57) **ABSTRACT**

A computer-implemented method for the encoding of, and computation on, distributions of data, the method comprising: obtaining a first set of data items; obtaining a second set of data items; generating a first tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the first set; generating a second tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the second set in which the parameters used to encode the distribution of the data items of the second set are the same as the parameters used to encode the distribution of the data items of the first set; generating a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding a probability distribution representing the result of applying an arithmetic operation on the first probability distribution and the second probability distribution; outputting the third tuple.



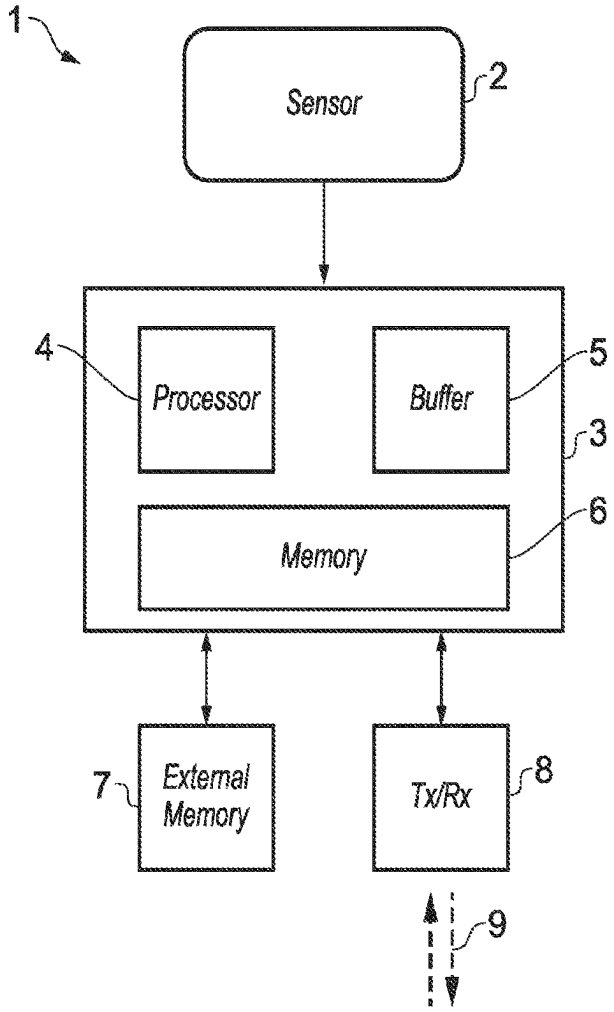

1
Sensor
2
4
Processor
Buffer
5
3
Memory
6
7
External Memory
Tx/Rx
8
9

FIG. 1

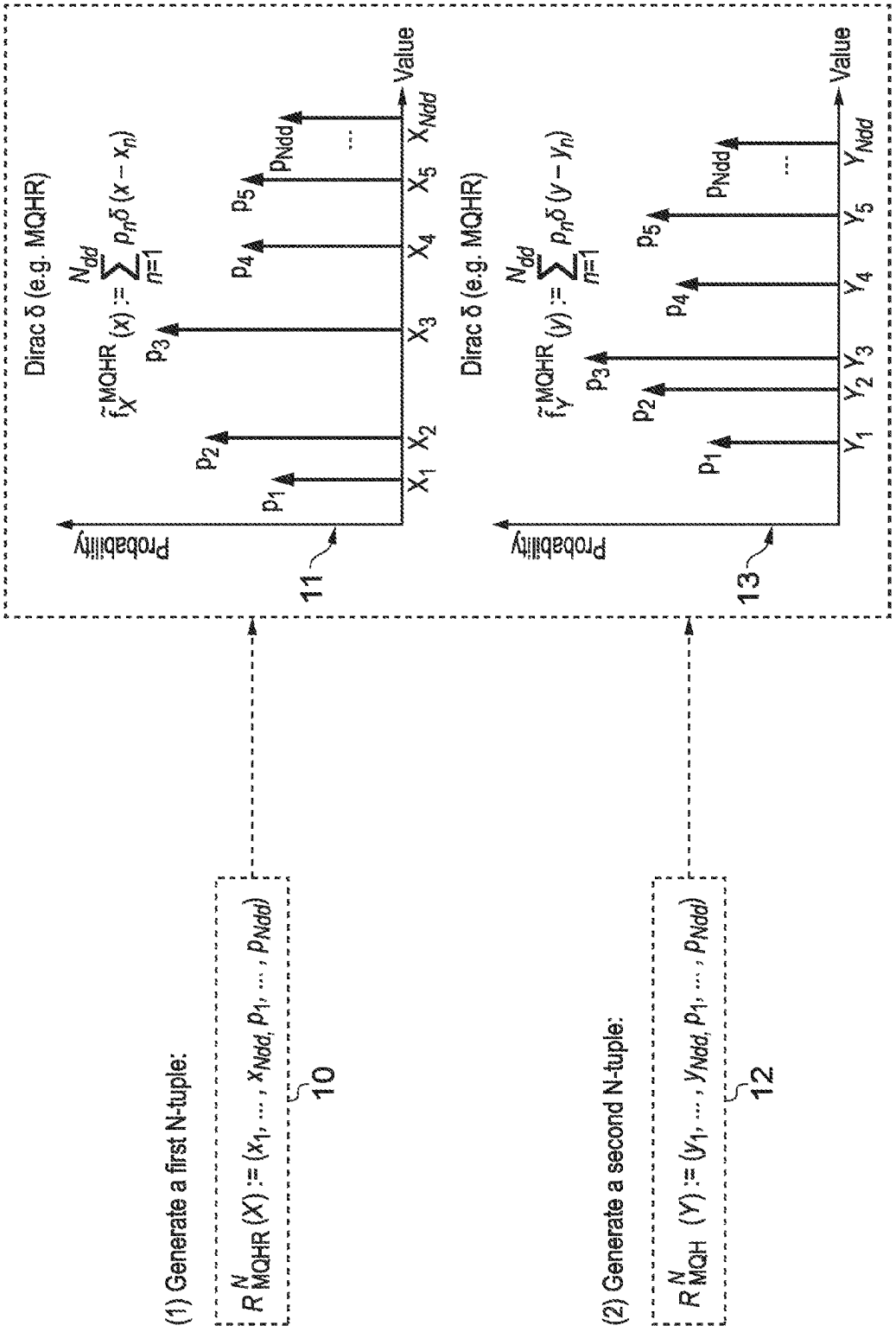

(1) Generate a first N-tuple:
$R^{N}_{MQHR}(X) := (x_1, \ldots, x_{Ndd}, p_1, \ldots, p_{Ndd})$
10
(2) Generate a second N-tuple:
$R^{N}_{MQH}(Y) := (y_1, \ldots, y_{Ndd}, p_1, \ldots, p_{Ndd})$
12
Dirac δ (e.g. MQHR)
$\tilde{f}^{MQHR}_{X}(x) := \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n)$
Probability
Value
p_1 p_2 p_3 p_4 p_5 p_{Ndd}
X_1 X_2 X_3 X_4 X_5 X_{Ndd}
11
Dirac δ (e.g. MQHR)
$\tilde{f}^{MQHR}_{Y}(y) := \sum_{n=1}^{N_{dd}} p_n \delta(y - y_n)$
Probability
Value
p_1 p_2 p_3 p_4 p_5 p_{Ndd}
Y_1 Y_2 Y_3 Y_4 Y_5 Y_{Ndd}
13

FIG. 2

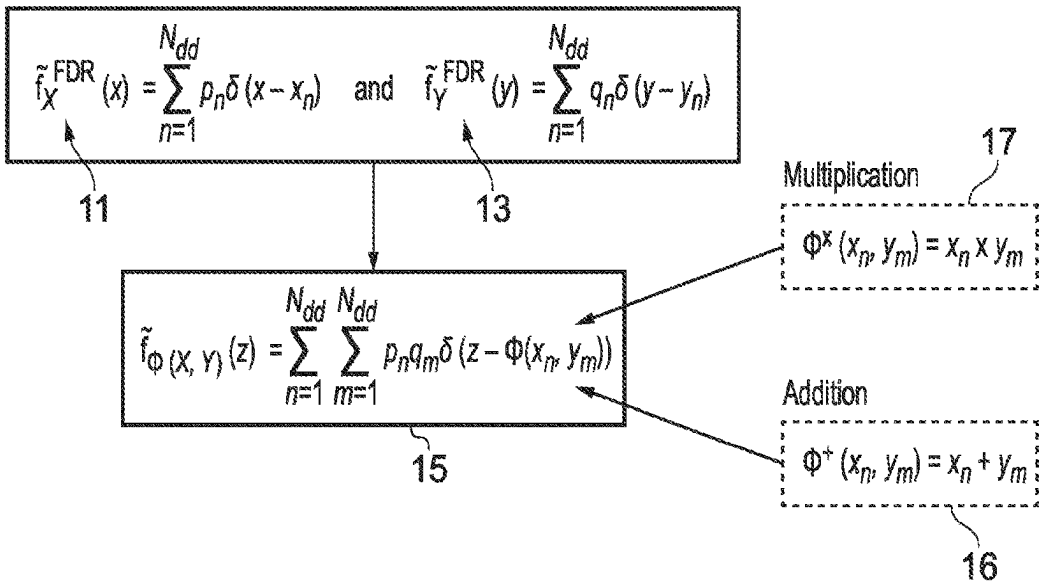

$\tilde{f}_X^{FDR}(x) = \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n)$ and $\tilde{f}_Y^{FDR}(y) = \sum_{n=1}^{N_{dd}} q_n \delta(y - y_n)$
11
13
17
Multiplication
$\Phi^x(x_n, y_m) = x_n \times y_m$
$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m))$
15
Addition
$\Phi^+(x_n, y_m) = x_n + y_m$
16

FIG. 3

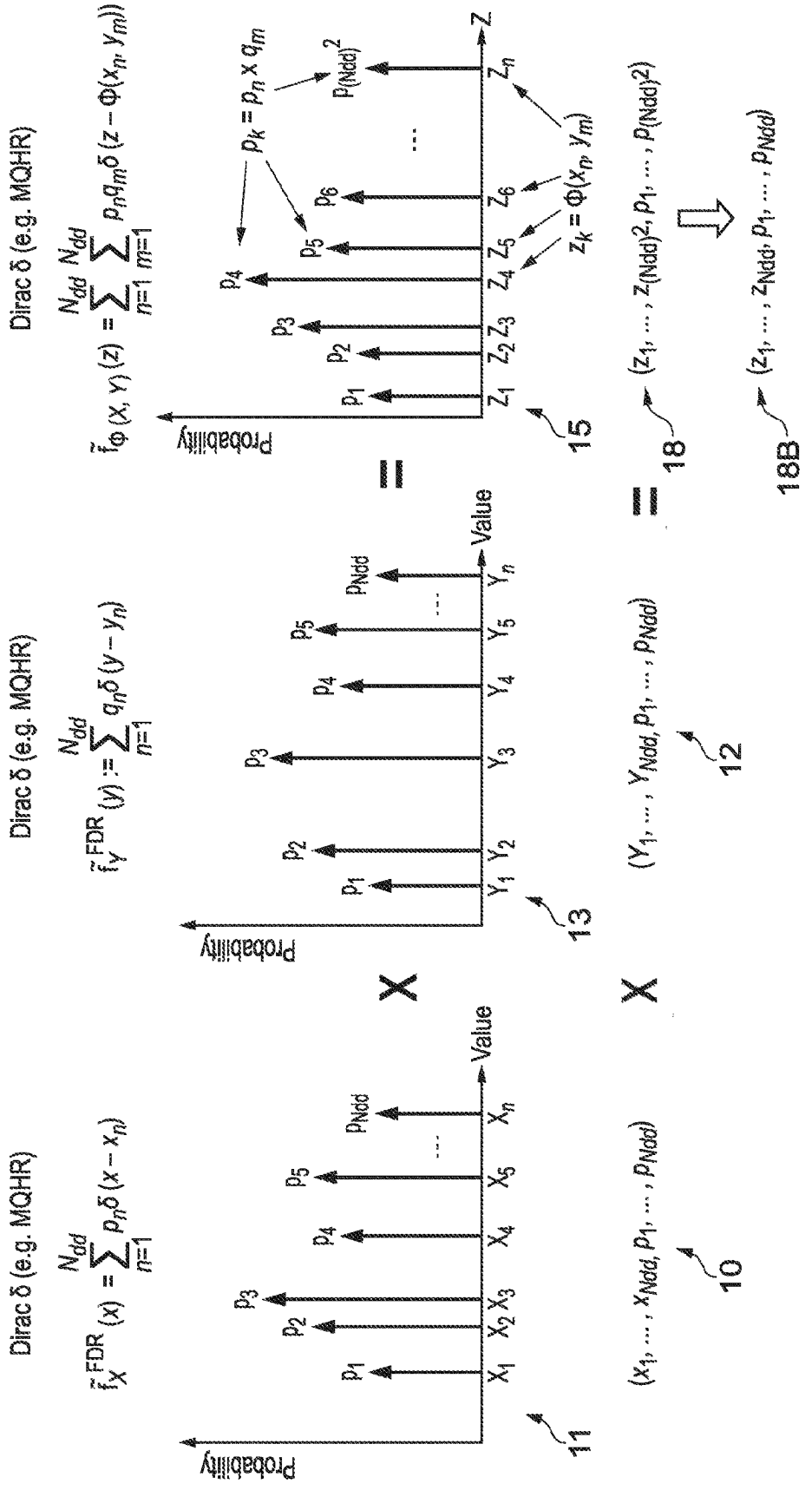

Dirac δ (e.g. MQHR)
Dirac δ (e.g. MQHR)
Dirac δ (e.g. MQHR)
Probability
Value
Z
11
12
13
15
10
18
18B
X
=

FIG. 4

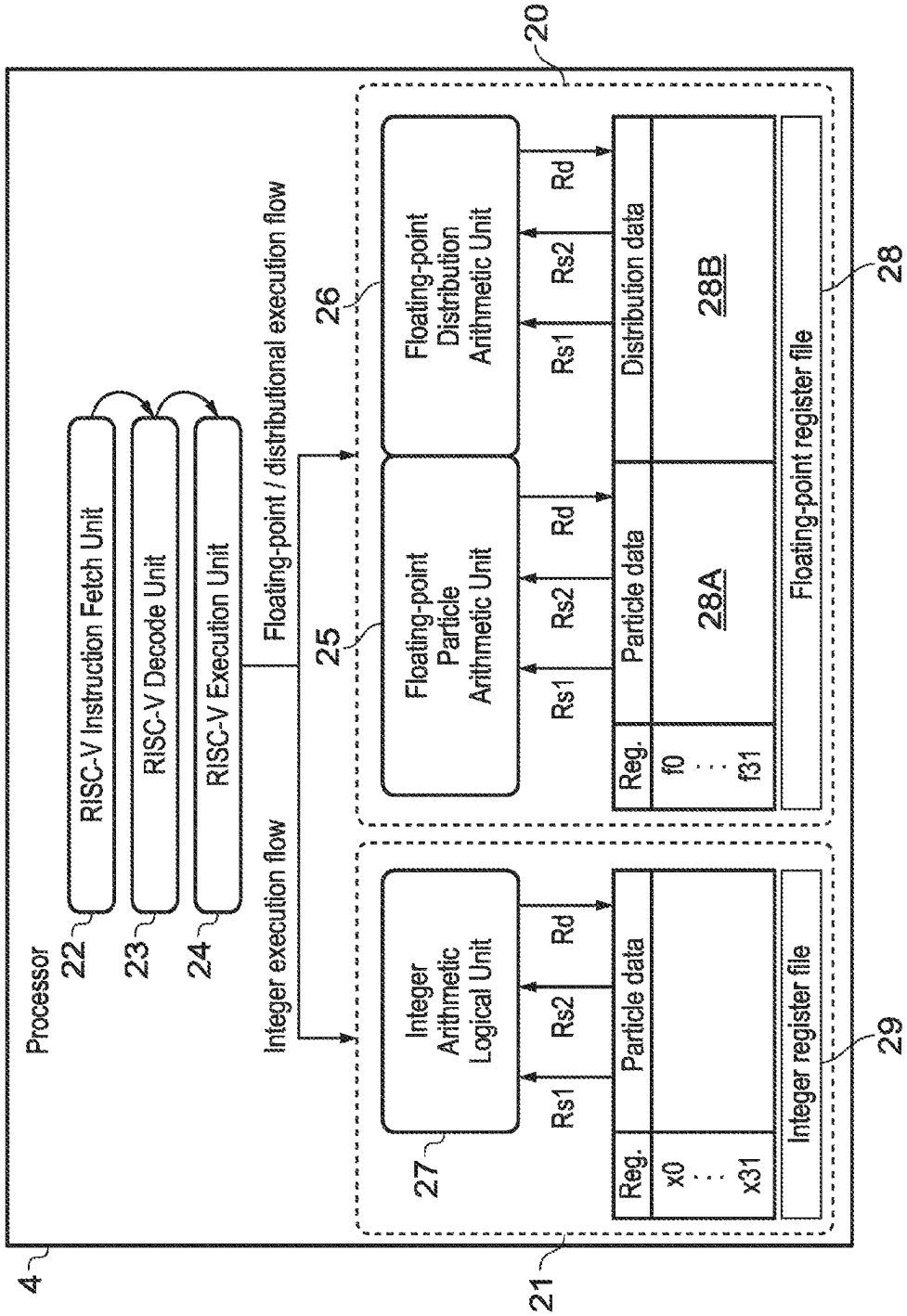

Processor
RISC-V Instruction Fetch Unit
RISC-V Decode Unit
RISC-V Execution Unit
22
23
24
Integer execution flow
Floating-point / distributional execution flow
25
26
20
Integer Arithmetic Logical Unit
27
Floating-point Particle Arithmetic Unit
Floating-point Distribution Arithmetic Unit
Rs1
Rs2
Rd
Reg.
Particle data
Distribution data
x0
x31
f0
f31
28A
28B
Integer register file
Floating-point register file
29
28
21
4

FIG. 5

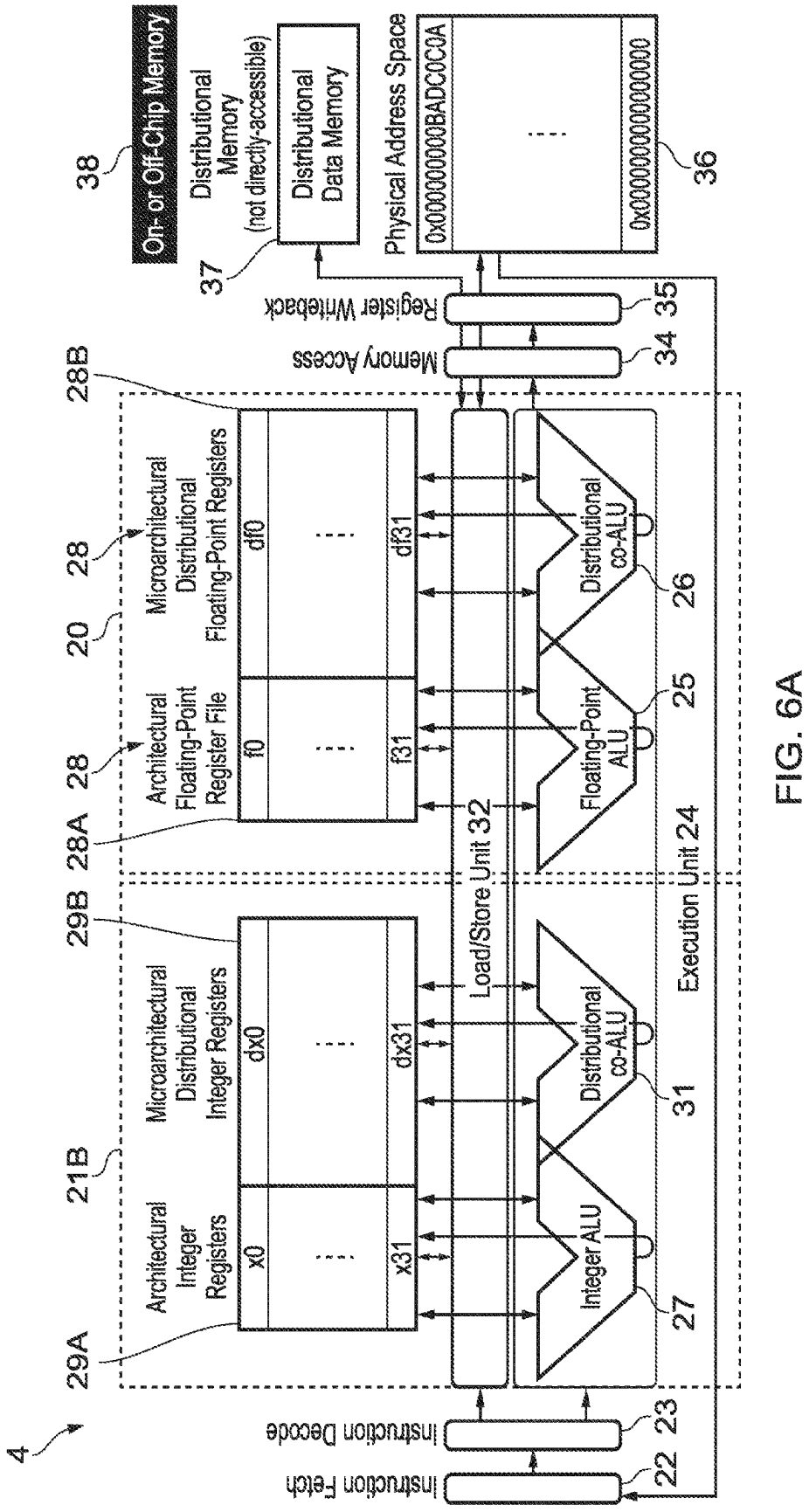
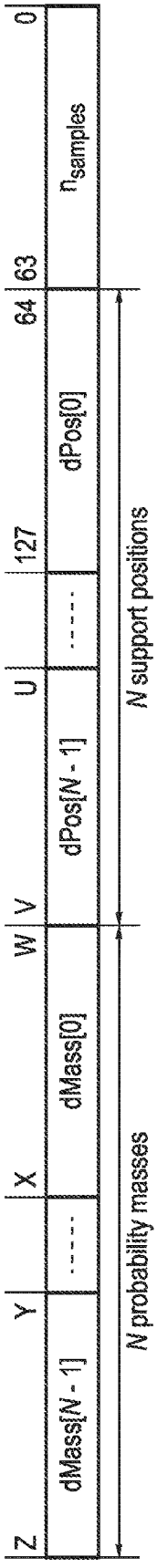

4
Instruction Fetch
22
Instruction Decode
23
29A
Architectural Integer Registers
x0
x31
21B
Microarchitectural Distributional Integer Registers
dx0
dx31
29B
28A
Architectural Floating-Point Register File
f0
f31
28
20
Microarchitectural Distributional Floating-Point Registers
df0
df31
28B
Load/Store Unit 32
Integer ALU
27
Distributional co-ALU
31
Floating-Point ALU
25
Distributional co-ALU
26
Execution Unit 24
Memory Access
34
Register Writeback
35
Physical Address Space
0x0000000000000000
0x000000000BADC0C0A
36
Distributional Data Memory
37
Distributional Memory (not directly-accessible)
On- or Off-Chip Memory
38

FIG. 6A



N: Size of the distributional representation. It contains *N* support positions and *N* probability masses, each 64 bits long

$n_{samples}$: The 64-bit long number of particle samples that Laplace used to generate the distributional representation

U: Beginning of the *N*th support position bitfield. Starts after (*N* - 1) positions of 64 bits plus 64 bits for $n_{samples}$. U=64*N*

V: End of the *N*th support position bitfield. V=U + 63 =64(*N* + 1) - 1 bits

W: Beginning of the 1st probability mass bitfield. W = V + 1 = 64(*N* + 1) bits

X: End of the 1st probability mass bitfield. X = W + 63 = 64(*N* + 2) - 1 bits

Y: Beginning of the *N*th probability mass bitfield. Starts after *N* positions and (*N* - 1) masses plus $n_{samples}$. Y = 2*N* * 64 bits

Z: End of the *N*th probability mass bitfield. Z =Y + 63 = 64(2*N* + 1) - 1 bits

FIG. 6B

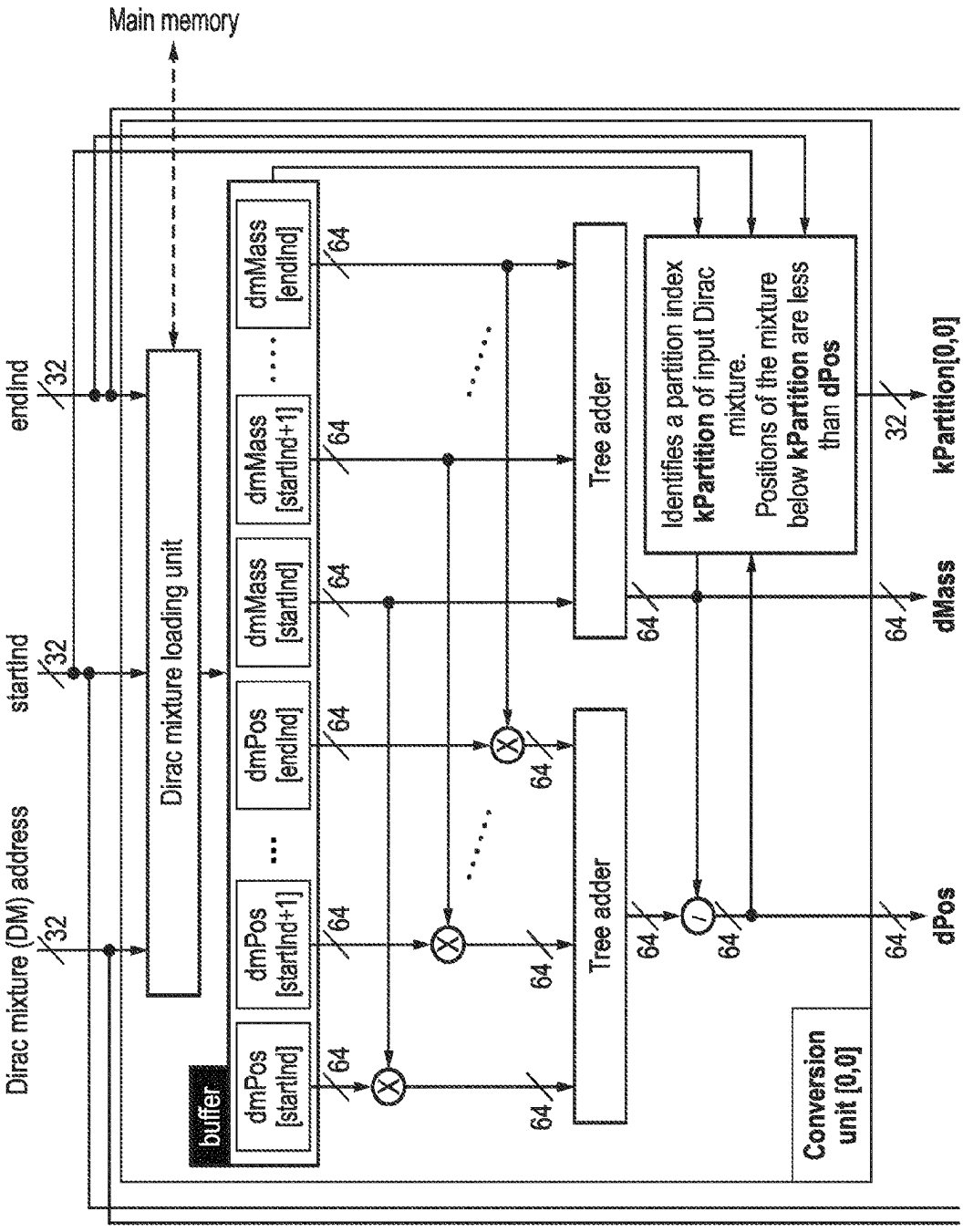

Main memory
endInd
32
startInd
32
Dirac mixture (DM) address
32
Dirac mixture loading unit
buffer
dmPos [startInd]
dmPos [startInd+1]
...
dmPos [endInd]
dmMass [startInd]
dmMass [startInd+1]
....
dmMass [endInd]
64
Tree adder
Tree adder
Identifies a partition index kPartition of input Dirac mixture.
Positions of the mixture below kPartition are less than dPos
Conversion unit [0,0]
kPartition[0,0]
dMass
dPos

FIG. 6C

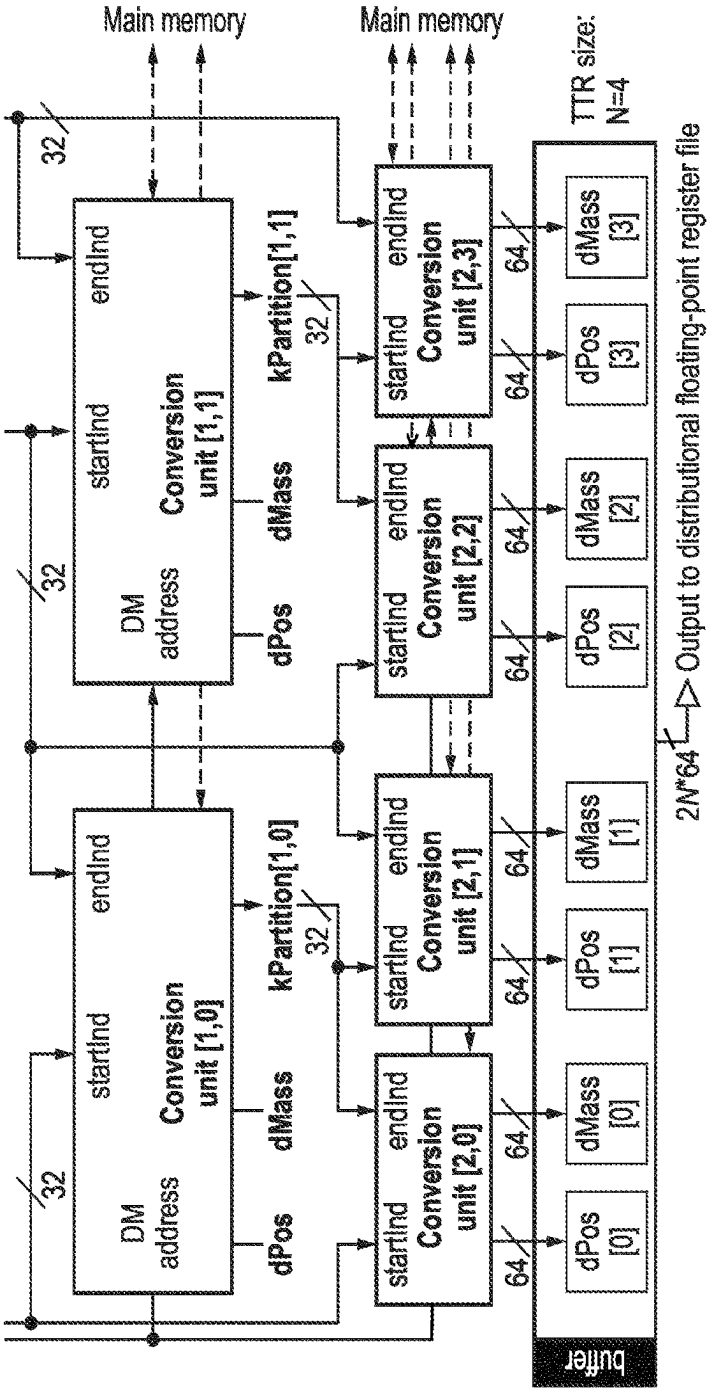
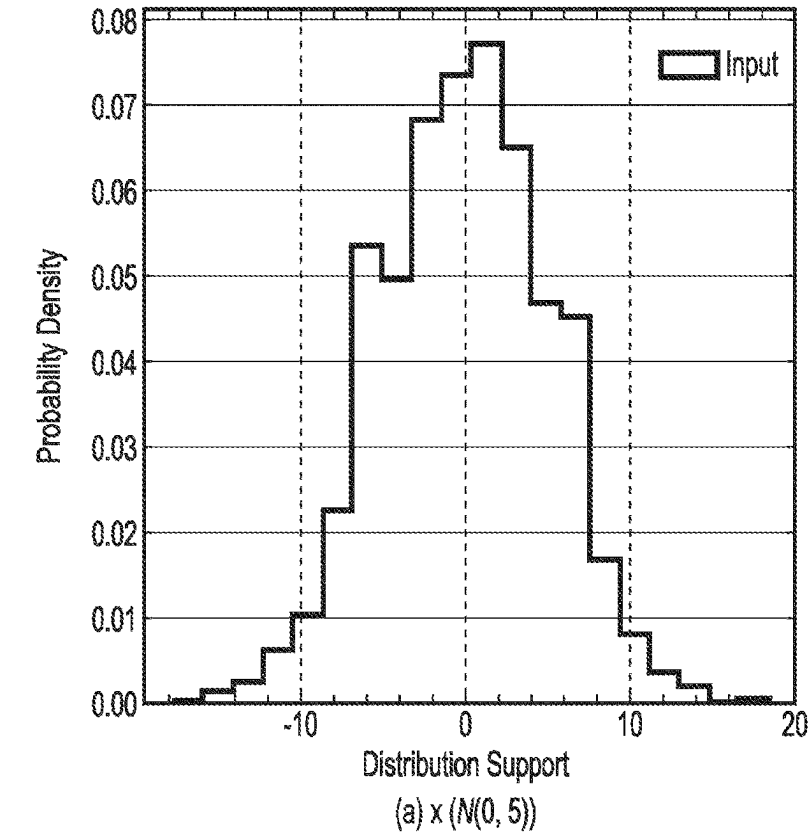


FIG. 6C (Continued)


Input
Probability Density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
-10
0
10
20
Distribution Support
(a) x (N(0, 5))

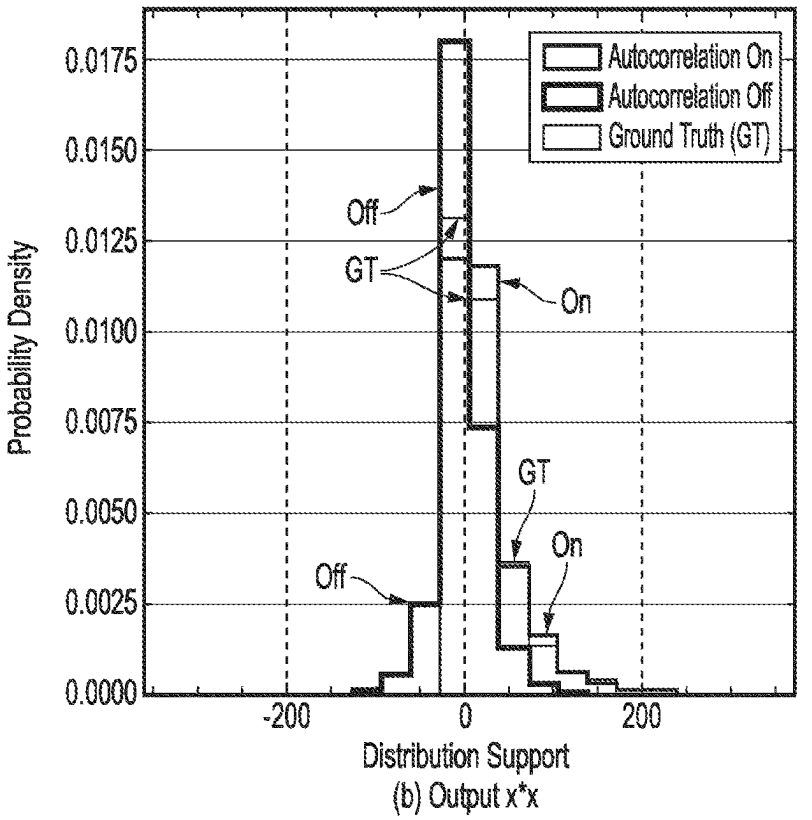

Autocorrelation On
Autocorrelation Off
Ground Truth (GT)
Off
GT
On
GT
On
Off
Probability Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
-200
0
200
Distribution Support
(b) Output x*x

FIG. 6D

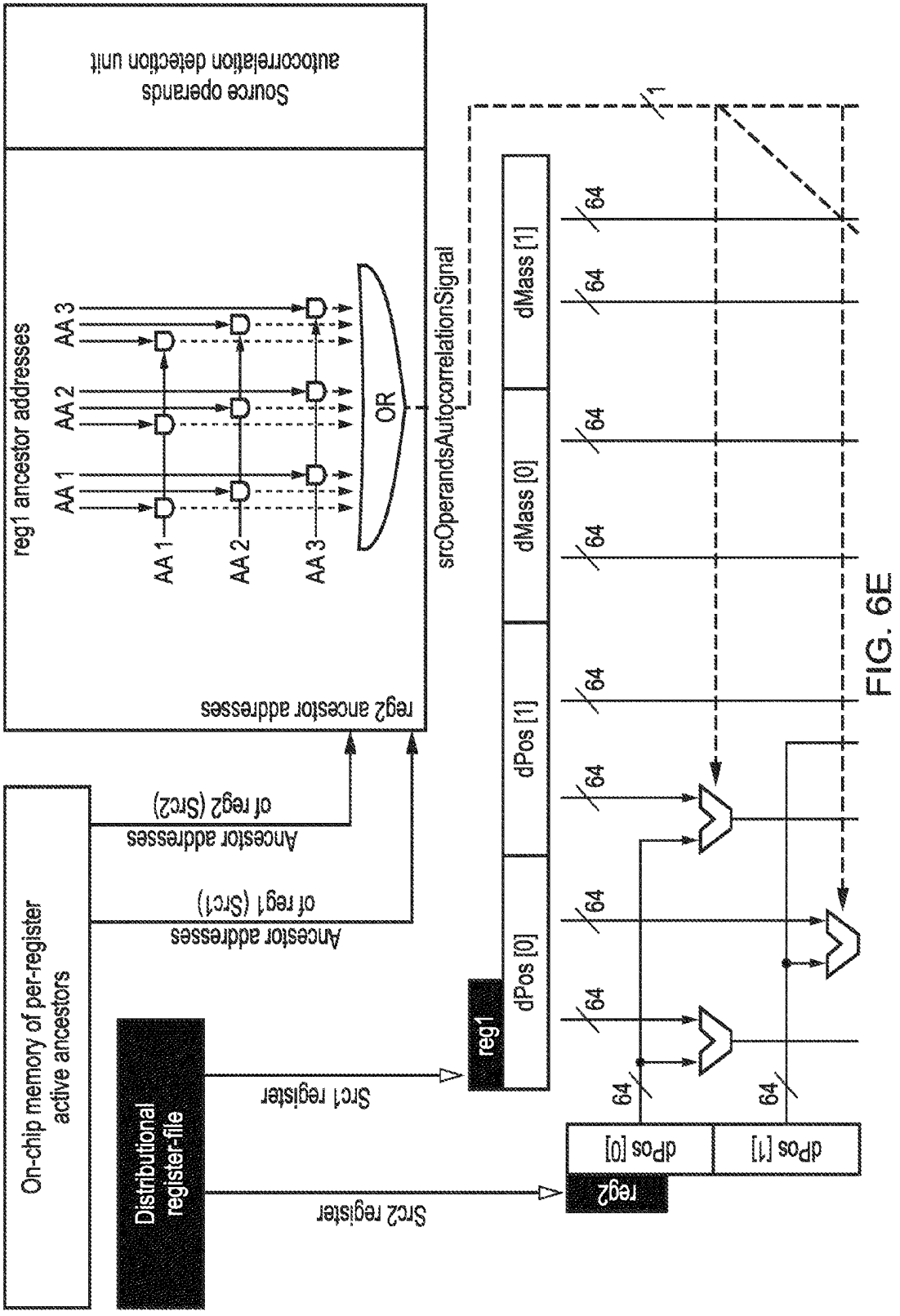

On-chip memory of per-register active ancestors
Distributional register-file
Src1 register
Src2 register
Ancestor addresses of reg1 (Src1)
Ancestor addresses of reg2 (Src2)
Source operands autocorrelation detection unit
reg1 ancestor addresses
reg2 ancestor addresses
AA 1
AA 2
AA 3
OR
srcOperandsAutocorrelationSignal
reg1
reg2
dPos [0]
dPos [1]
dMass [0]
dMass [1]
64
1

FIG. 6E

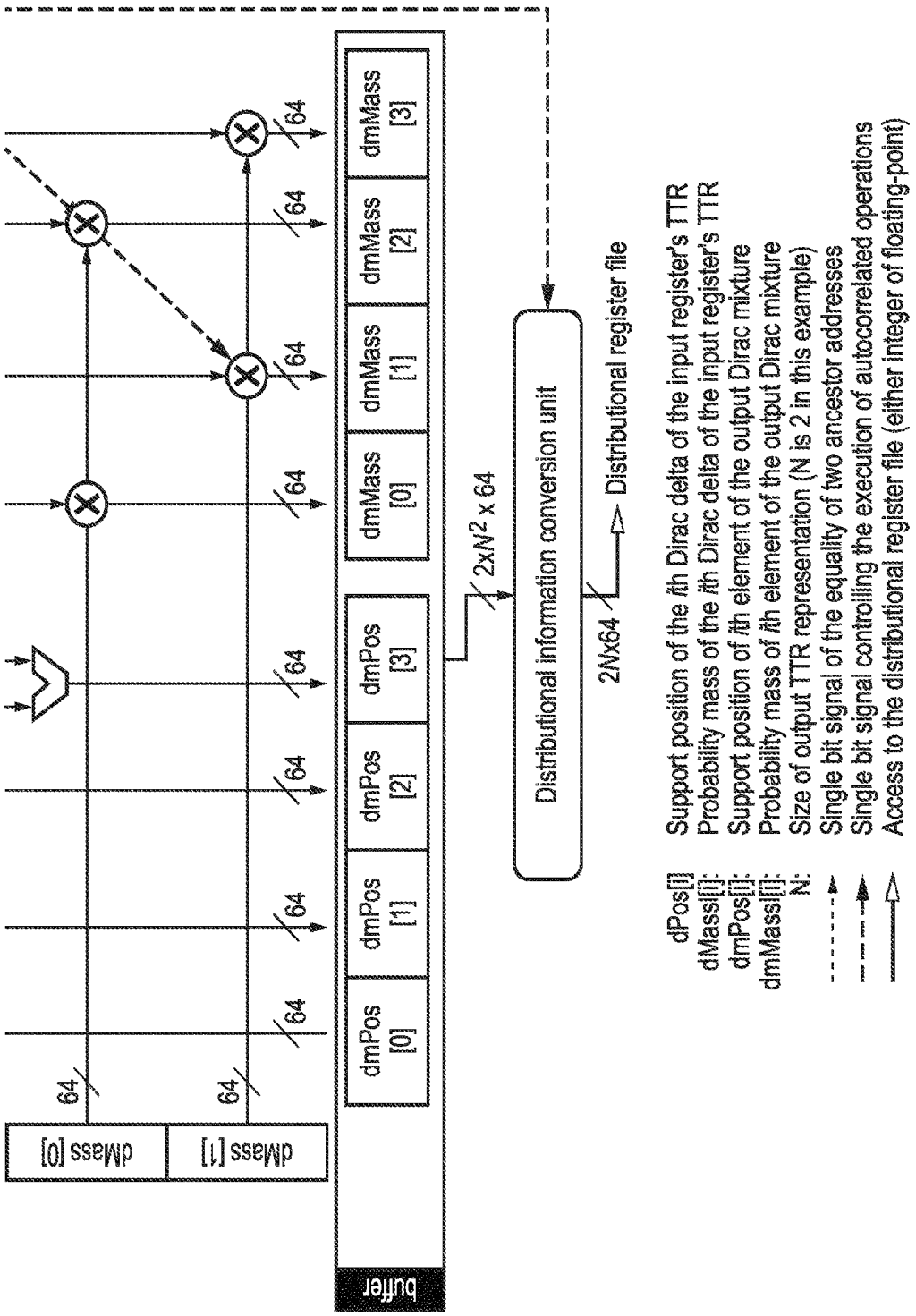


FIG. 6E (Continued)

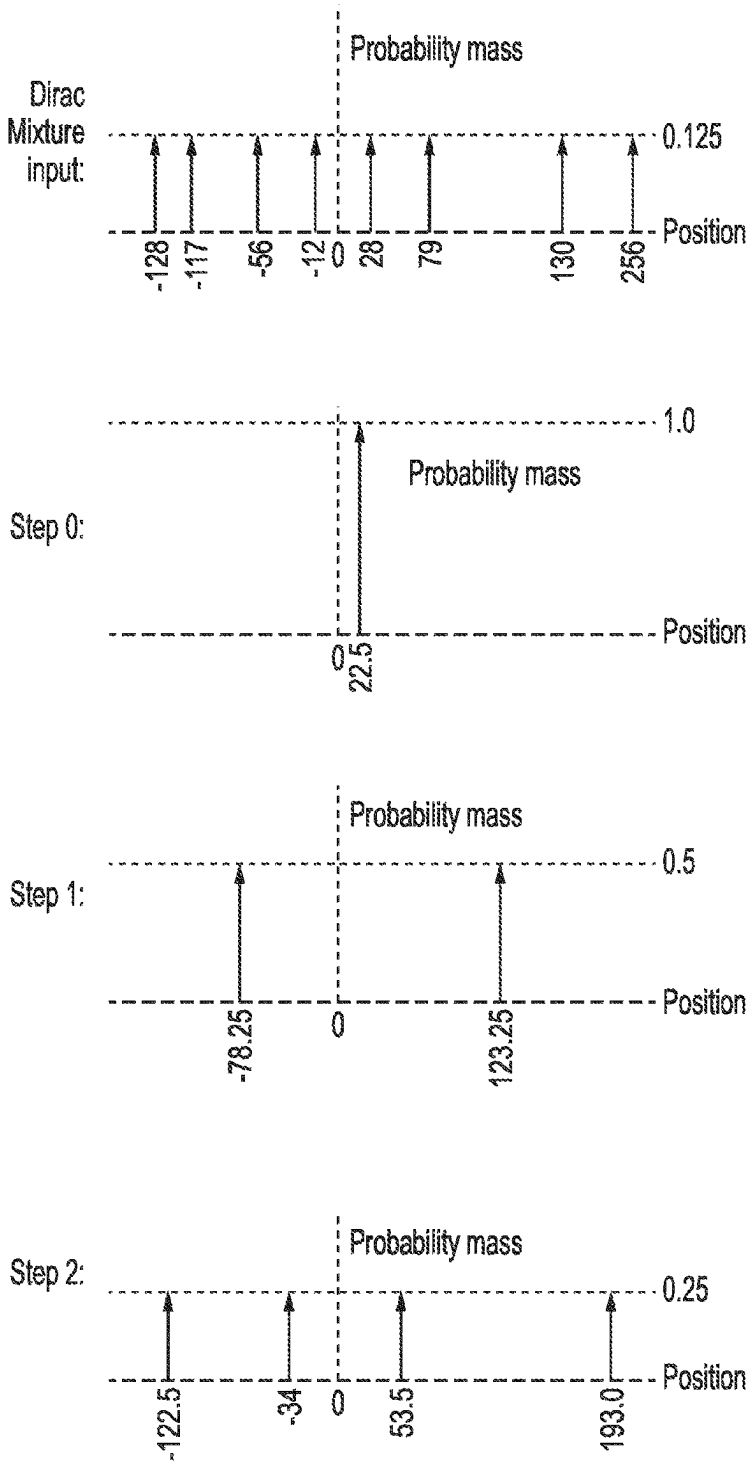

Probability mass
Dirac Mixture input:
0.125
Position
-128
-117
-56
-12
0
28
79
130
256
1.0
Probability mass
Step 0:
Position
0
22.5
Probability mass
Step 1:
0.5
Position
-78.25
0
123.25
Probability mass
Step 2:
0.25
Position
-122.5
-34
0
53.5
193.0

FIG. 7

Row	Instruction address	Dest. reg.	List of origin addresses (before)	List of origin addresses (after)	Comments
1st iteration (i=2)					
0	8004184	fa4	0x0	-72(s0)	
1	8004188	fa5	0x0	-72(s0)	
2	800418c	fa5	-72(s0)	-72(s0)	Auto-correlation calculation triggered
3	8004190	fa4	-72(s0)	-48(s0)	
4	8004194	fa5	-72(s0)	-72(s0) → -48(s0)	
5	8004198	fa5	-72(s0) → -48(s0)	-72(s0) → -48(s0)	Origin address list of fa5 stored in -48(s0)
2nd iteration (i=4)					
6	8004184	fa4	-48(s0)	-72(s0)	
7	8004188	fa5	-72(s0) → -48(s0)	-72(s0)	
8	800418c	fa5	-72(s0)	-72(s0)	Auto-correlation calculation triggered
9	8004190	fa4	-72(s0)	-72(s0) → -48(s0)	
10	8004194	fa5	-72(s0)	-72(s0) → -48(s0)	Auto-correlation calculation triggered
11	8004198	fa5	-72(s0) → -48(s0)	-72(s0) → -48(s0)	Origin address list of fa5 stored in -48(s0)

FIG. 8

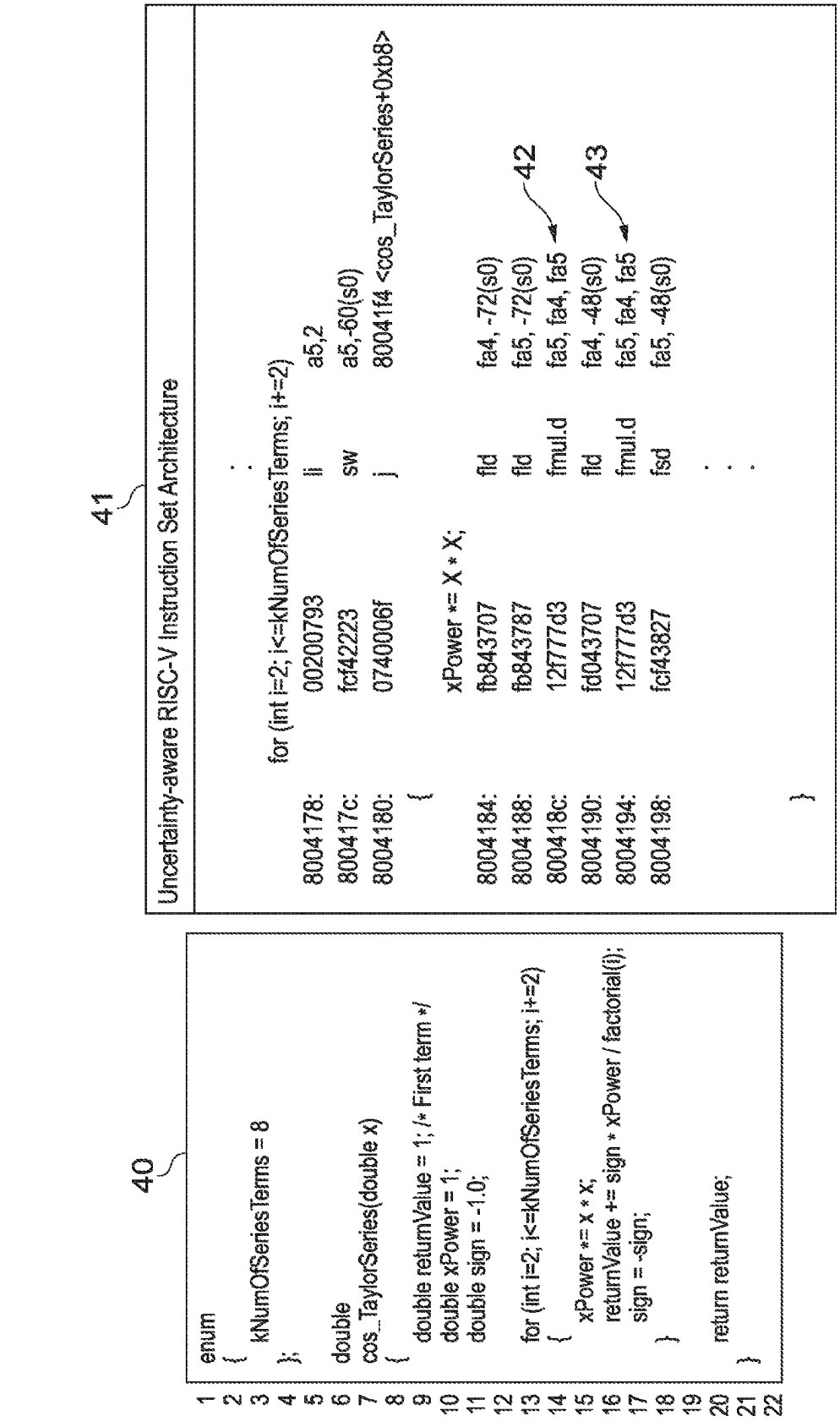

40
41
Uncertainty-aware RISC-V Instruction Set Architecture
enum
{
kNumOfSeriesTerms = 8
};
double
cos_TaylorSeries(double x)
{
double returnValue = 1; /* First term */
double xPower = 1;
double sign = -1.0;
for (int i=2; i<=kNumOfSeriesTerms; i+=2)
{
xPower *= x * x;
returnValue += sign * xPower / factorial(i);
sign = -sign;
}
return returnValue;
}
. . .
for (int i=2; i<=kNumOfSeriesTerms; i+=2)
8004178: 00200793 li a5,2
800417c: fcf42223 sw a5,-60(s0)
8004180: 0740006f j 8004184 <cos_TaylorSeries+0xb8>
{
xPower *= x * x;
8004184: fb843707 fld fa4, -72(s0)
8004188: fb843787 fld fa5, -72(s0)
800418c: 12f777d3 fmul.d fa5, fa4, fa5
8004190: fd043707 fld fa4, -48(s0)
8004194: 12f777d3 fmul.d fa5, fa4, fa5
8004198: fcf43827 fsd fa5, -48(s0)
. . .
}
42
43

FIG. 9

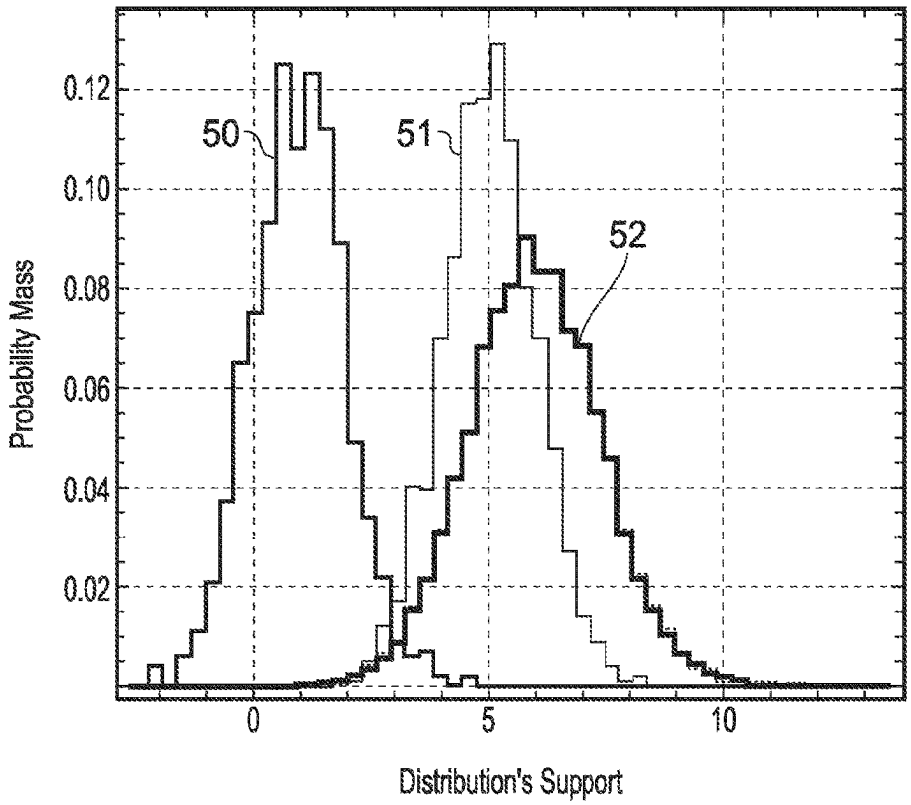

0.12
0.10
0.08
0.06
0.04
0.02
Probability Mass
50
51
52
0
5
10
Distribution's Support

FIG. 10

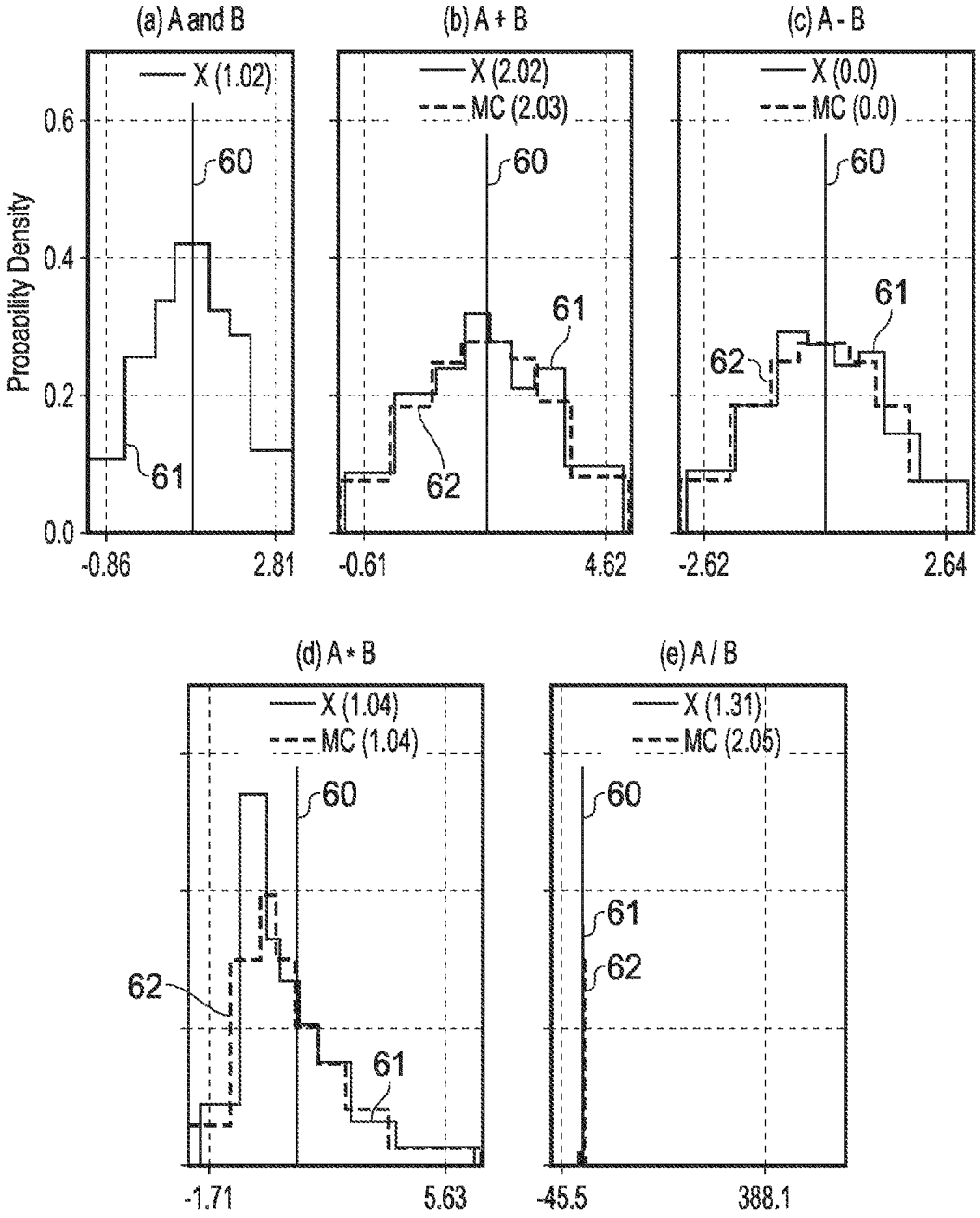

(a) A and B
X (1.02)
60
61
Probability Density
0.6
0.4
0.2
0.0
-0.86
2.81
(b) A + B
X (2.02)
MC (2.03)
60
61
62
-0.61
4.62
(c) A - B
X (0.0)
MC (0.0)
60
61
62
-2.62
2.64
(d) A * B
X (1.04)
MC (1.04)
60
62
61
-1.71
5.63
(e) A / B
X (1.31)
MC (2.05)
60
61
62
-45.5
388.1

FIG. 11

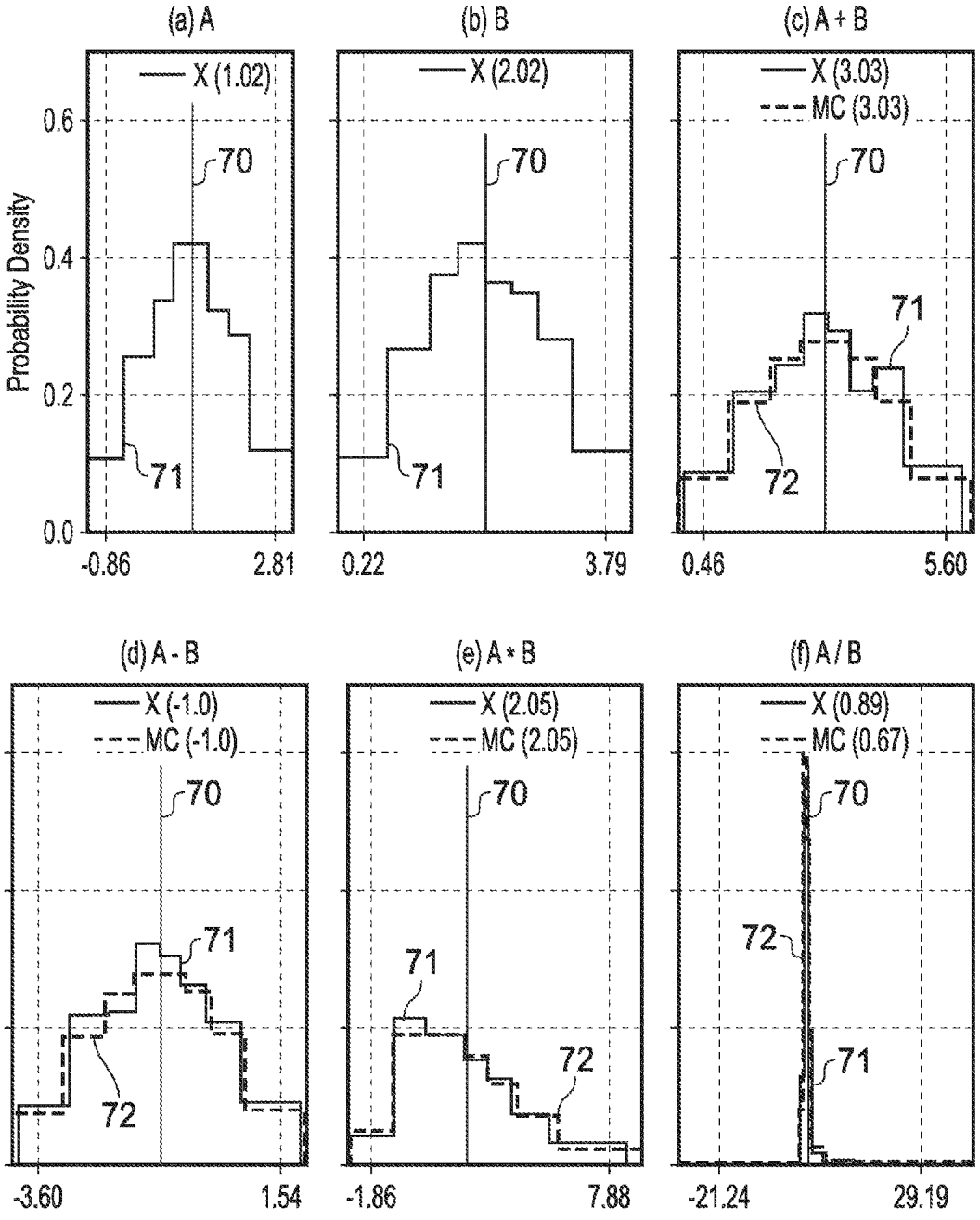

(a) A
X (1.02)
70
71
Probability Density
0.6
0.4
0.2
0.0
-0.86
2.81
(b) B
X (2.02)
70
71
0.22
3.79
(c) A + B
X (3.03)
MC (3.03)
70
71
72
0.46
5.60
(d) A - B
X (-1.0)
MC (-1.0)
70
71
72
-3.60
1.54
(e) A * B
X (2.05)
MC (2.05)
70
71
72
-1.86
7.88
(f) A / B
X (0.89)
MC (0.67)
70
72
71
-21.24
29.19

FIG. 12

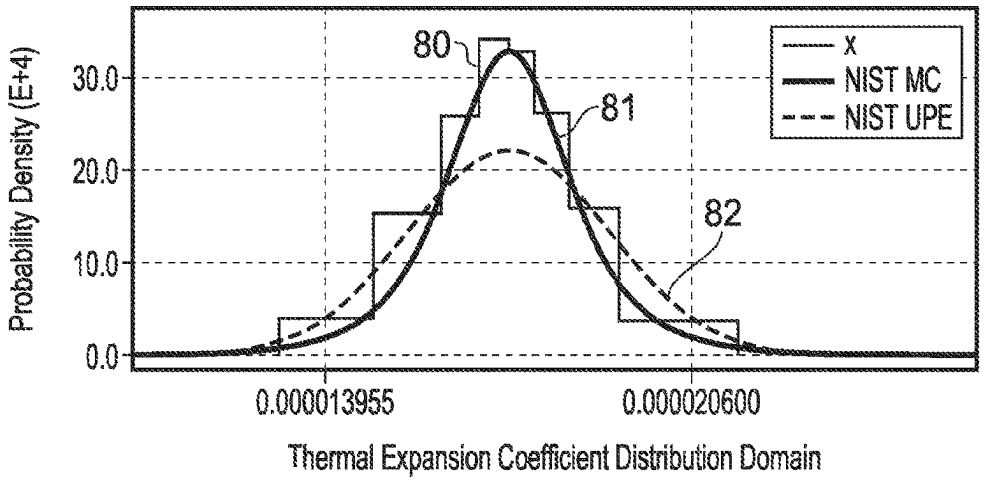


FIG. 13A

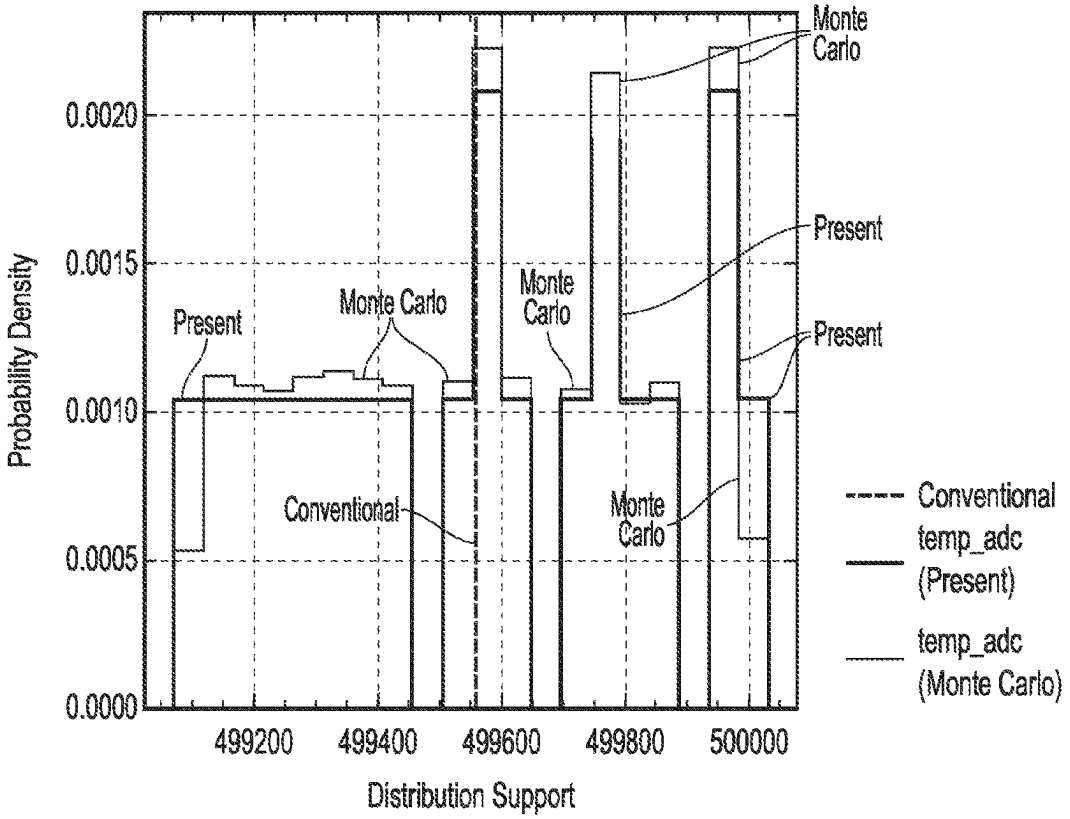


FIG. 13B: ADC value (TTR-64)

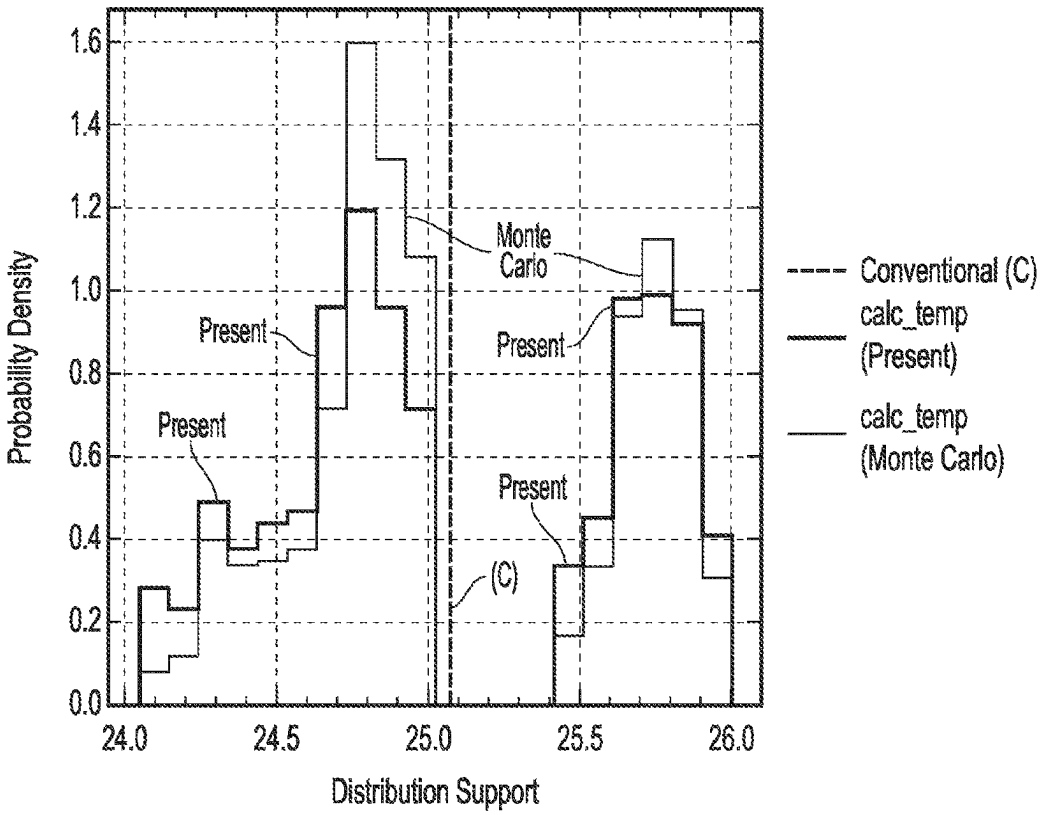


FIG. 13C: Temperature (TTR-64)

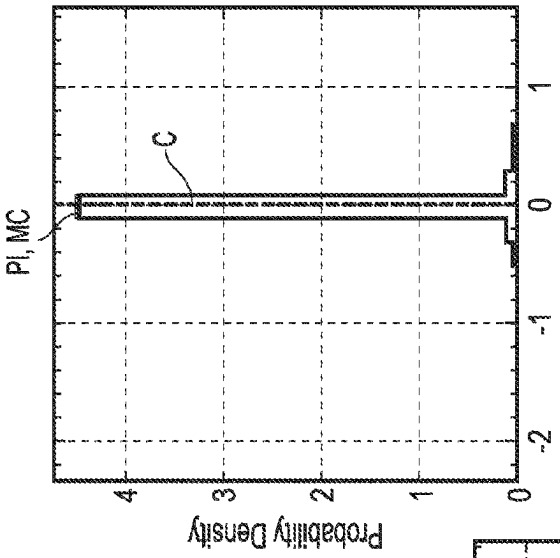


FIG. 13F: Phi (Iteration 5)

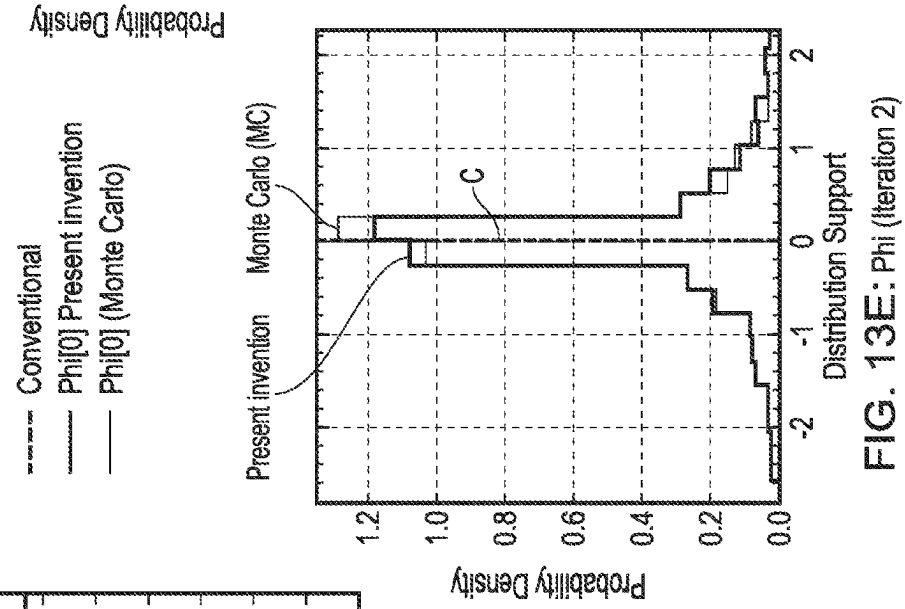


FIG. 13E: Phi (Iteration 2)

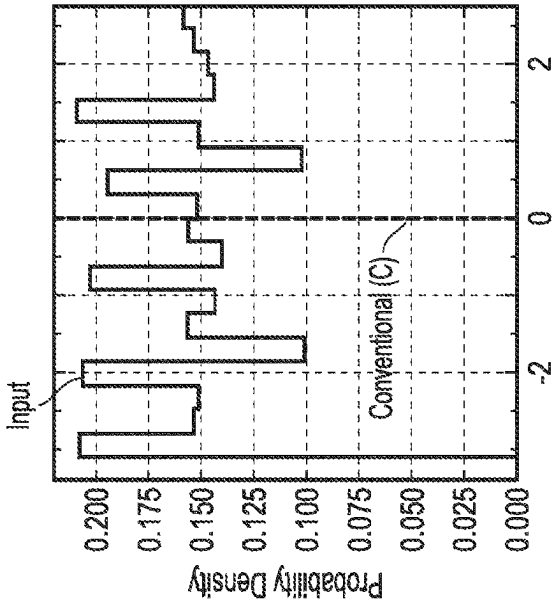
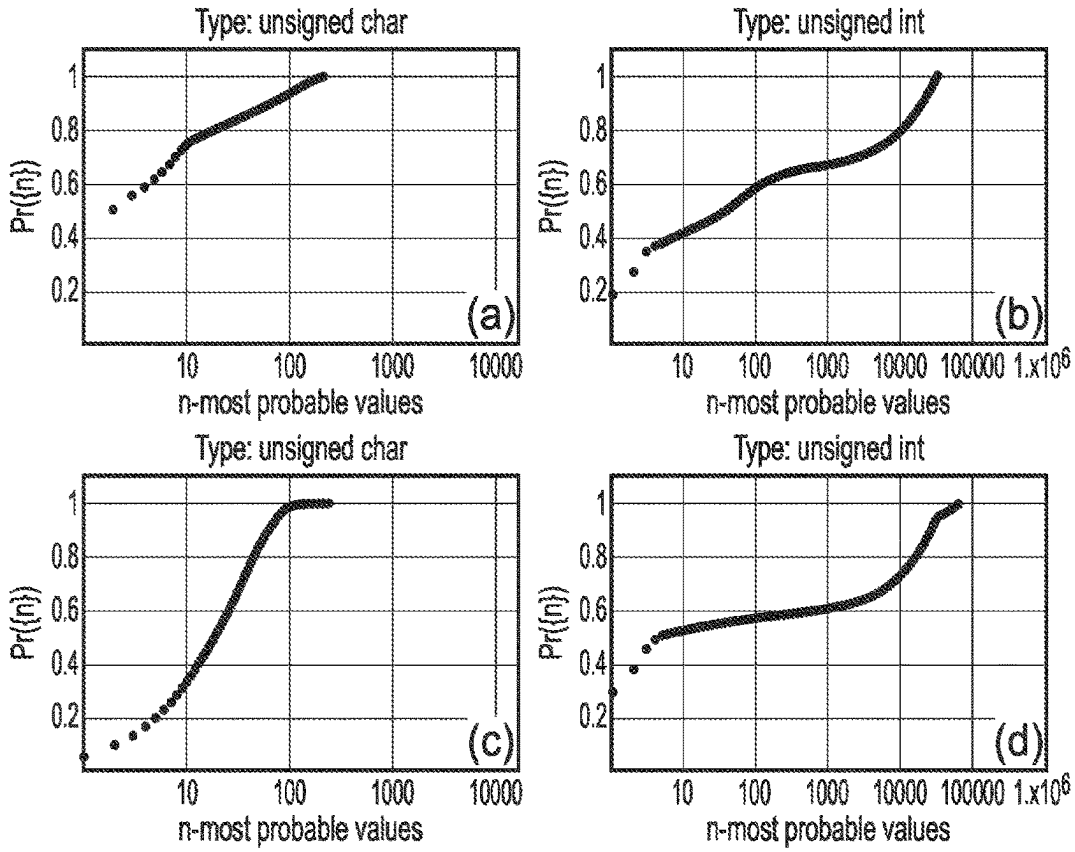


FIG. 13D: Phi (Iteration 0)


Type: unsigned char
Pr({n})
n-most probable values
(a)
Type: unsigned int
(b)
Type: unsigned char
(c)
Type: unsigned int
(d)
10
100
1000
10000
100000
1.x10^6
0.2
0.4
0.6
0.8
1

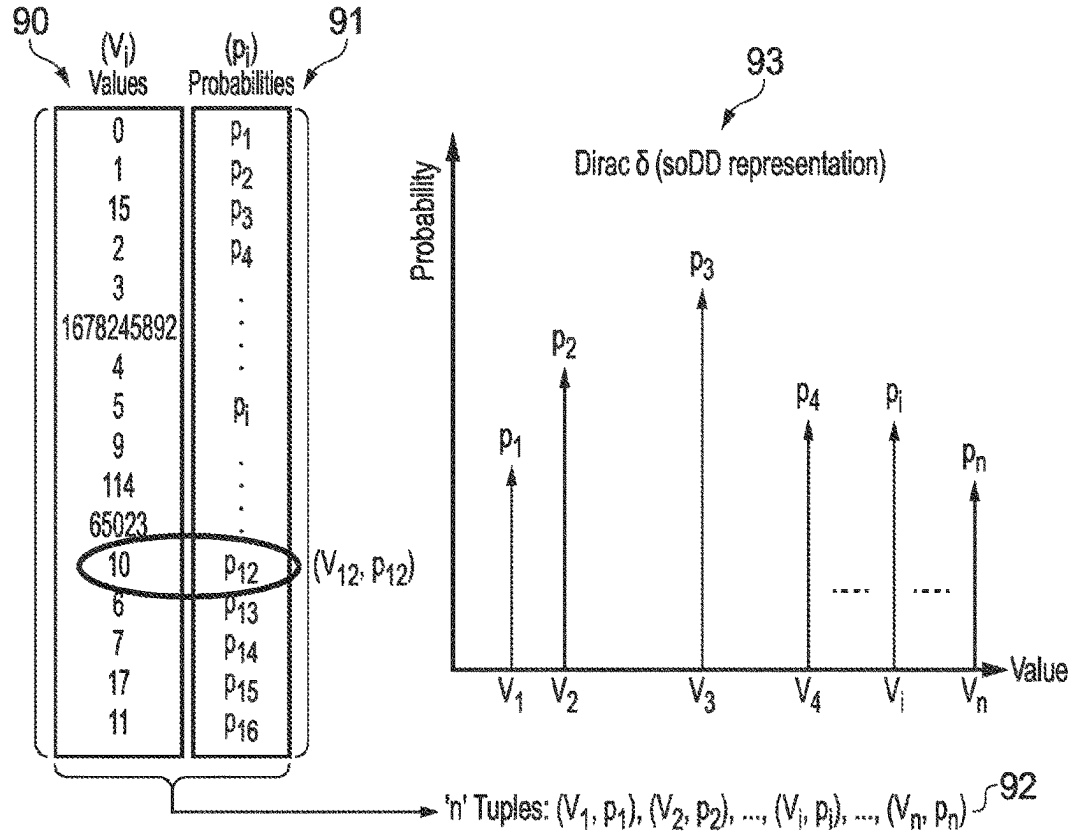

90
(Vi) Values
(pi) Probabilities
91
0
1
15
2
3
1678245892
4
5
9
114
65023
10
6
7
17
11
p1
p2
p3
p4
pi
p12
p13
p14
p15
p16
(V12, p12)
93
Dirac δ (soDD representation)
Probability
Value
V1
V2
V3
V4
Vi
Vn
pn
'n' Tuples: (V1, p1), (V2, p2), ..., (Vi, pi), ..., (Vn, pn)
92

FIG. 14

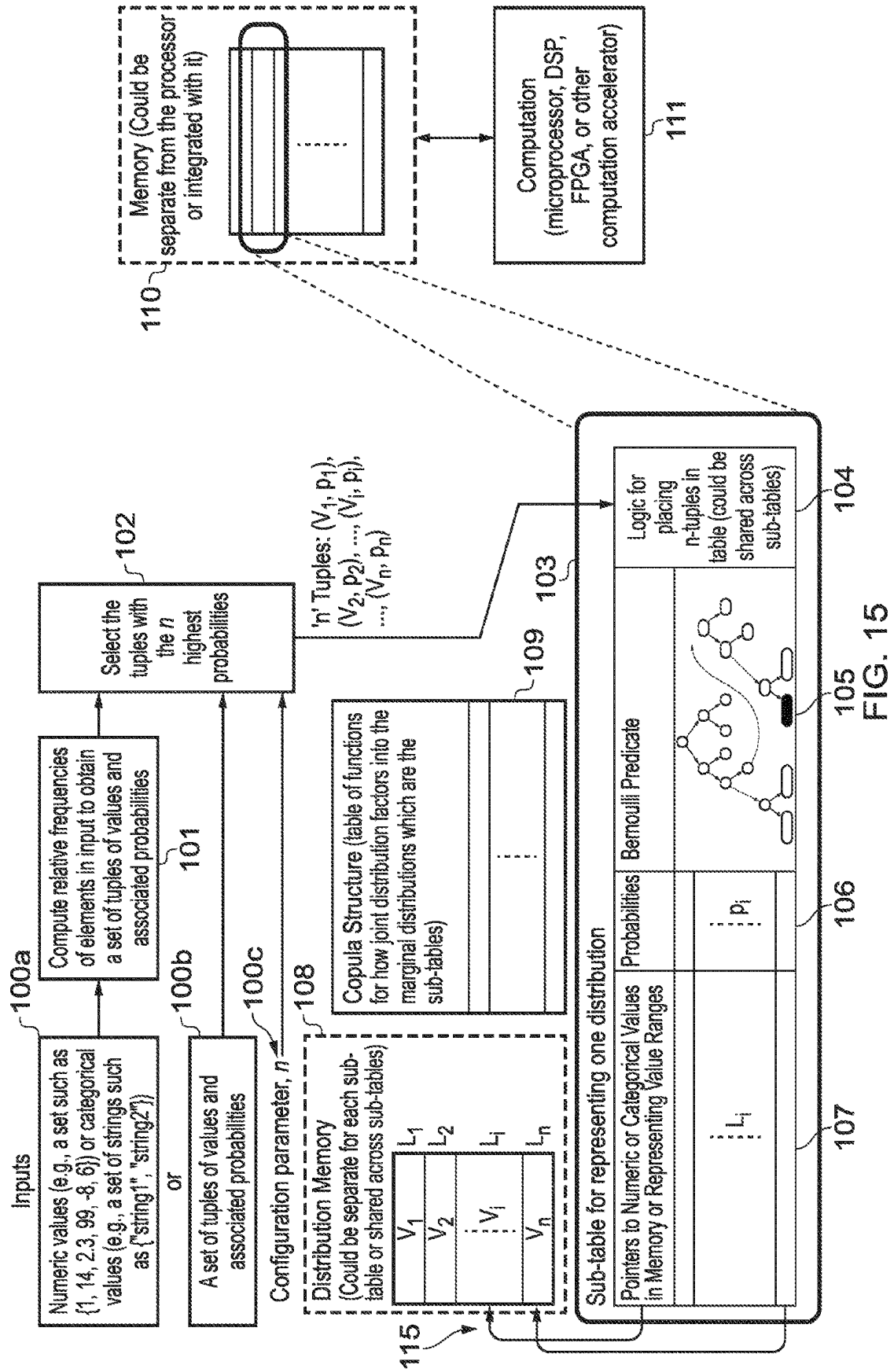

Inputs
100a
Numeric values (e.g., a set such as {1, 14, 2.3, 99, -8, 6}) or categorical values (e.g., a set of strings such as {"string1", "string2"})
or
A set of tuples of values and associated probabilities
100b
Compute relative frequencies of elements in input to obtain a set of tuples of values and associated probabilities
101
100c
Configuration parameter, n
108
Select the tuples with the n highest probabilities
102
'n' Tuples: (V1, P1), (V2, P2), ..., (Vi, Pi), ..., (Vn, Pn)
103
Distribution Memory
(Could be separate for each sub-table or shared across sub-tables)
V1 L1
V2 L2
Vi Li
Vn Ln
115
Copula Structure (table of functions for how joint distribution factors into the marginal distributions which are the sub-tables)
109
Sub-table for representing one distribution
Pointers to Numeric or Categorical Values in Memory or Representing Value Ranges
Probabilities
Bernoulli Predicate
Li
pi
106
107
105
Logic for placing n-tuples in table (could be shared across sub-tables)
104
Memory (Could be separate from the processor or integrated with it)
110
Computation (microprocessor, DSP, FPGA, or other computation accelerator)
111

FIG. 15

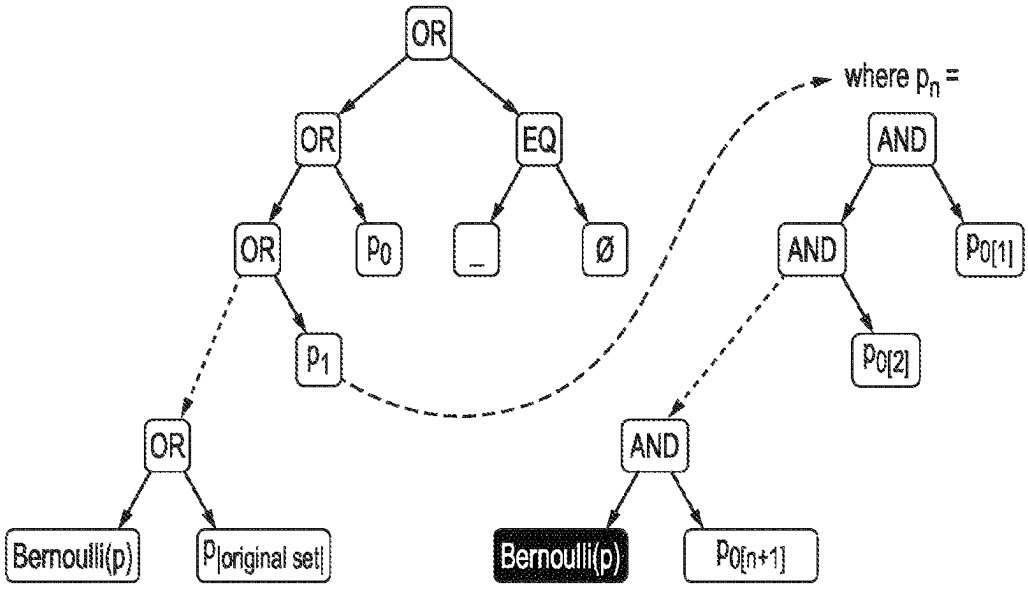

OR
OR
EQ
OR
p0
_
Ø
p1
OR
Bernoulli(p)
p|original set|
where pn =
AND
AND
p0[1]
p0[2]
AND
Bernoulli(p)
p0[n+1]

FIG. 16

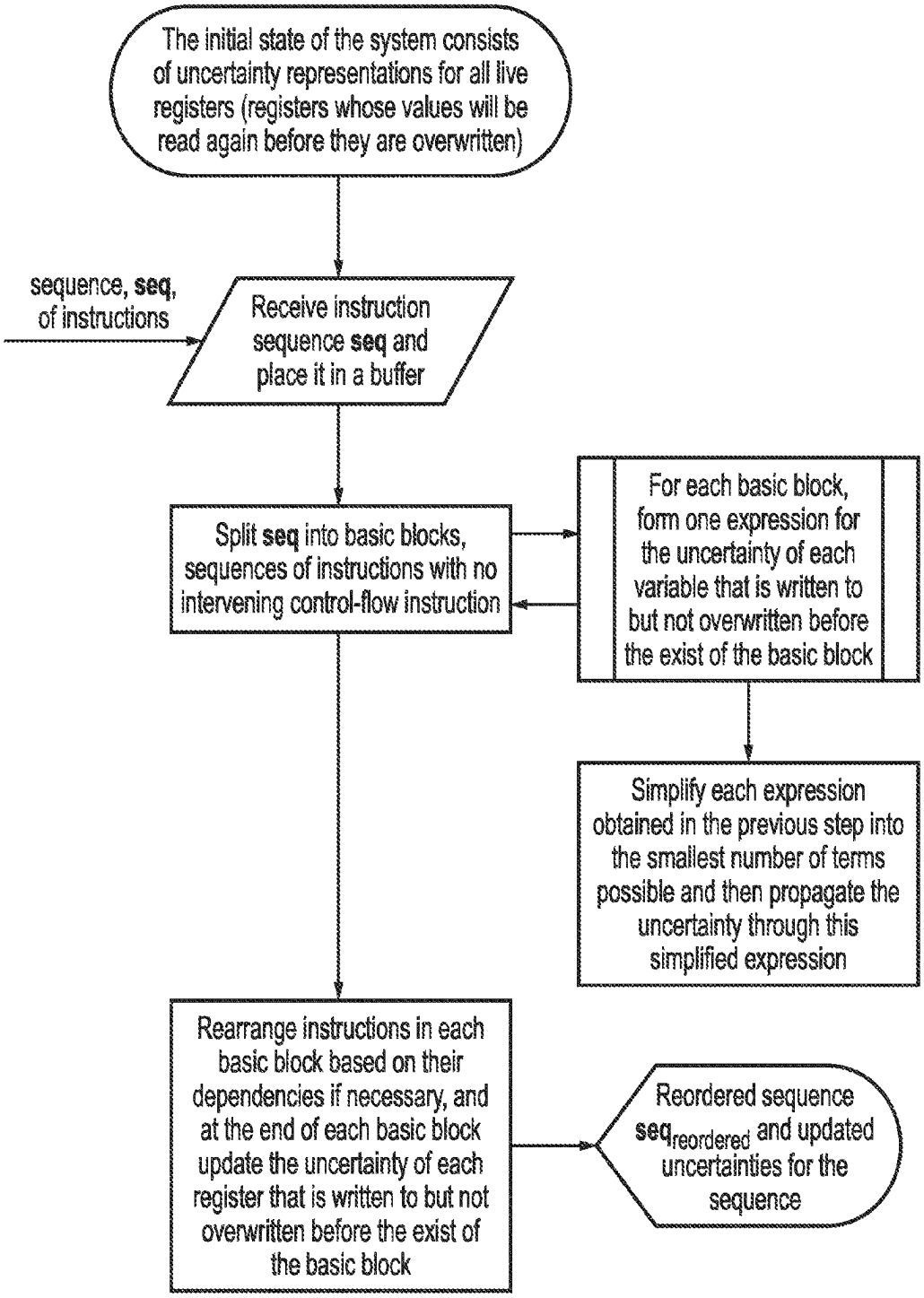

The initial state of the system consists of uncertainty representations for all live registers (registers whose values will be read again before they are overwritten)
sequence, seq, of instructions
Receive instruction sequence seq and place it in a buffer
Split seq into basic blocks, sequences of instructions with no intervening control-flow instruction
For each basic block, form one expression for the uncertainty of each variable that is written to but not overwritten before the exist of the basic block
Simplify each expression obtained in the previous step into the smallest number of terms possible and then propagate the uncertainty through this simplified expression
Rearrange instructions in each basic block based on their dependencies if necessary, and at the end of each basic block update the uncertainty of each register that is written to but not overwritten before the exist of the basic block
Reordered sequence seq reordered and updated uncertainties for the sequence

FIG. 17

(a)

```
float
f(float x1, float x2, float x3) {
        float  result;

        result = (x1 + x2) / (x1 - x2);
        result /= (x3*x3);

        return result;
}
```

(b)

```
0:   fd010113   addi    sp,sp,-48
4:   02812623   sw      s0,44(sp)
8:   03010413   addi    s0,sp,48
c:   fca42e27   fsw     fa0,-36(s0)
10:  fcb42c27   fsw     fa1,-40(s0)
14:  fcc42a27   fsw     fa2,-44(s0)
18:  fdc42707   flw     fa4,-36(s0)
1c:  fd842787   flw     fa5,-40(s0)
20:  00f77753   fadd.s  fa4,fa4,fa5
24:  fdc42687   flw     fa3,-36(s0)
28:  fd842787   flw     fa5,-40(s0)
2c:  08f6f7d3   fsub.s  fa5,fa3,fa5
30:  18f777d3   fdiv.s  fa5,fa4,fa5
34:  fef42627   fsw     fa5,-20(s0)
38:  fd442707   flw     fa4,-44(s0)
3c:  fd442787   flw     fa5,-44(s0)
40:  10f777d3   fmul.s  fa5,fa4,fa5
44:  fec42707   flw     fa4,-20(s0)
48:  18f777d3   fdiv.s  fa5,fa4,fa5
4c:  fef42627   fsw     fa5,-20(s0)
50:  fec42787   flw     fa5,-20(s0)
54:  20f78553   fmv.s   fa0, fa5
58:  02c12403   lw      s0,44(sp)
5c:  03010113   addi    sp,sp,48
60:  00008067   ret
```

FIG. 18

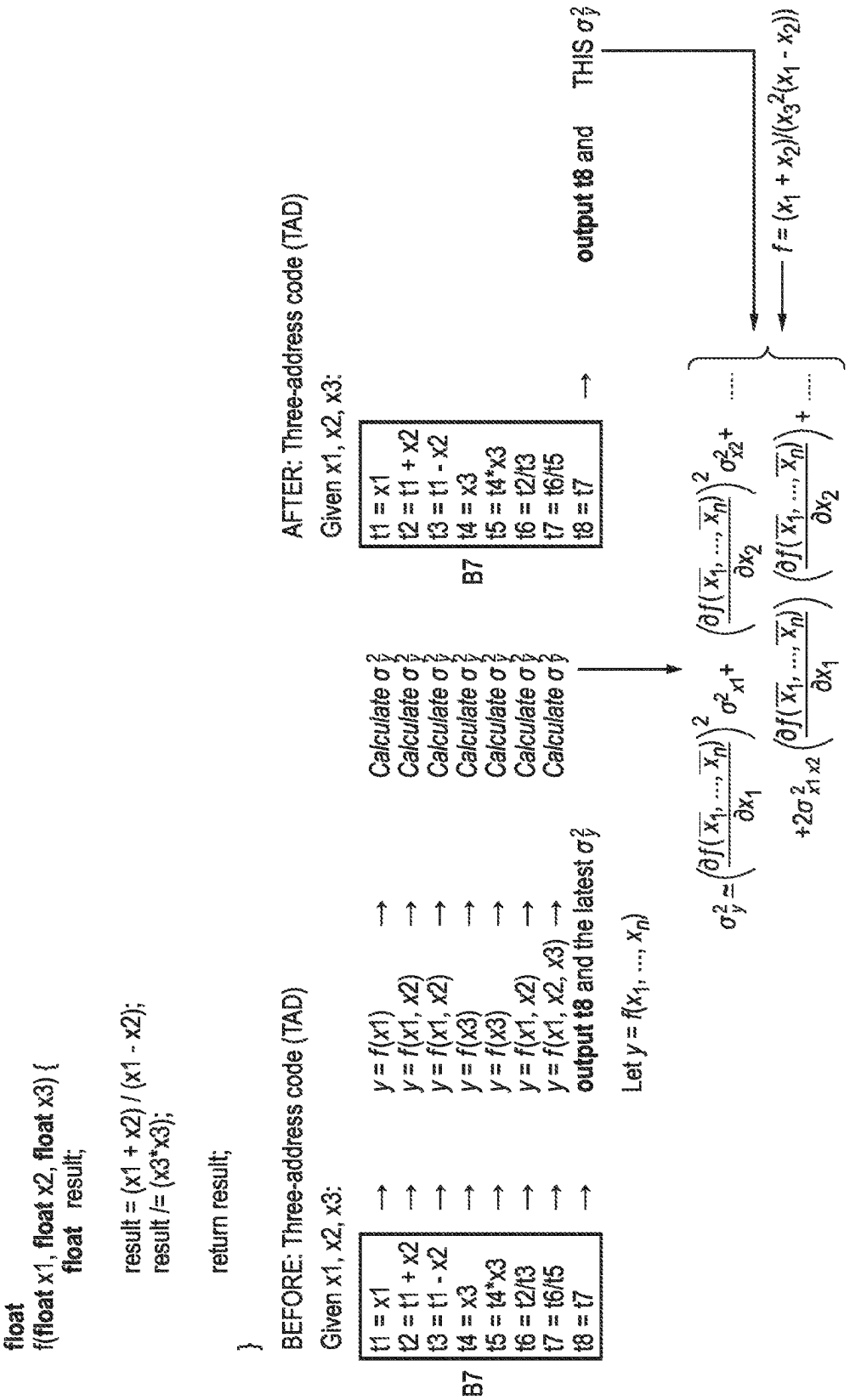

float
f(float x1, float x2, float x3) {
float result;
result = (x1 + x2) / (x1 - x2);
result /= (x3*x3);
return result;
}
BEFORE: Three-address code (TAD)
Given x1, x2, x3:
B7
t1 = x1
t2 = t1 + x2
t3 = t1 - x2
t4 = x3
t5 = t4*x3
t6 = t2/t3
t7 = t6/t5
t8 = t7
y = f(x1)
y = f(x1, x2)
y = f(x1, x2)
y = f(x3)
y = f(x3)
y = f(x1, x2)
y = f(x1, x2, x3)
output t8 and the latest σ²y
Let y = f(x1, ..., xn)
Calculate σ²y
AFTER: Three-address code (TAD)
Given x1, x2, x3:
B7
t1 = x1
t2 = t1 + x2
t3 = t1 - x2
t4 = x3
t5 = t4*x3
t6 = t2/t3
t7 = t6/t5
t8 = t7
output t8 and THIS σ²y
σ²y ≈ (∂f(x̄1, ..., x̄n)/∂x1)² σ²x1 + (∂f(x̄1, ..., x̄n)/∂x2)² σ²x2 + ...
+2σ²x1 x2 (∂f(x̄1, ..., x̄n)/∂x1)(∂f(x̄1, ..., x̄n)/∂x2) + ...
f = (x1 + x2)/(x3²(x1 - x2))

FIG. 19

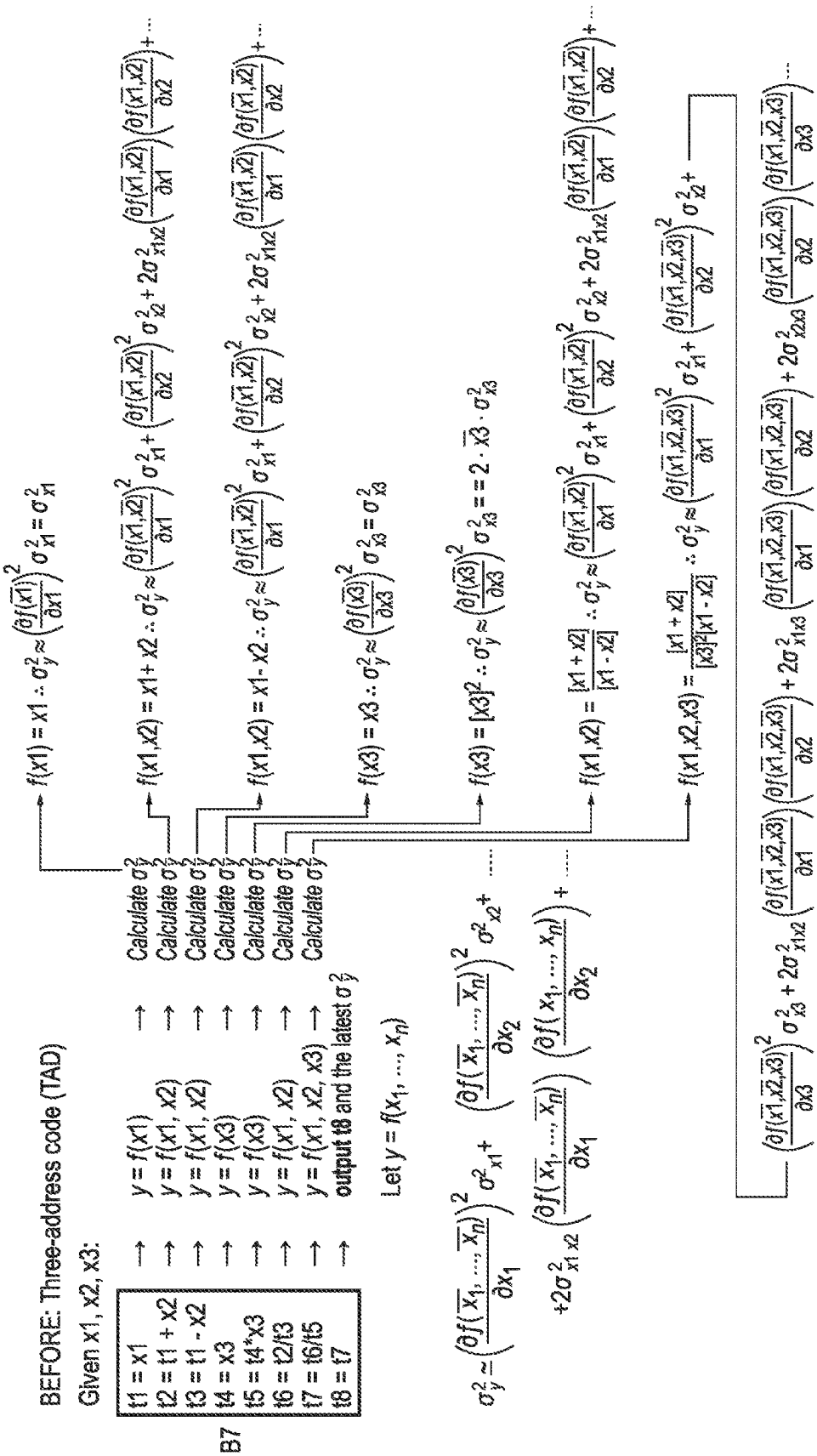

BEFORE: Three-address code (TAD)
Given x1, x2, x3:
B7
t1 = x1
t2 = t1 + x2
t3 = t1 - x2
t4 = x3
t5 = t4*x3
t6 = t2/t3
t7 = t6/t5
t8 = t7
y = f(x1)
y = f(x1, x2)
y = f(x1, x2)
y = f(x3)
y = f(x3)
y = f(x1, x2)
y = f(x1, x2, x3)
output t8 and the latest σ²y
Let y = f(x1, ..., xn)
Calculate σ²y
f(x1) = x1 ∴ σ²y ≈ (∂f(x1)/∂x1)² σ²x1 = σ²x1
f(x1,x2) = x1 + x2 ∴ σ²y ≈ (∂f(x1,x2)/∂x1)² σ²x1 + (∂f(x1,x2)/∂x2)² σ²x2 + 2σ²x1x2 (∂f(x1,x2)/∂x1)(∂f(x1,x2)/∂x2) + ...
f(x1,x2) = x1 - x2 ∴ σ²y ≈ (∂f(x1,x2)/∂x1)² σ²x1 + (∂f(x1,x2)/∂x2)² σ²x2 + 2σ²x1x2 (∂f(x1,x2)/∂x1)(∂f(x1,x2)/∂x2) + ...
f(x3) = x3 ∴ σ²y ≈ (∂f(x3)/∂x3)² σ²x3 = σ²x3
f(x3) = [x3]² ∴ σ²y ≈ (∂f(x3)/∂x3)² σ²x3 = = 2 · x3 · σ²x3
f(x1,x2) = [x1 + x2]/[x1 - x2] ∴ σ²y ≈ (∂f(x1,x2)/∂x1)² σ²x1 + (∂f(x1,x2)/∂x2)² σ²x2 + 2σ²x1x2 (∂f(x1,x2)/∂x1)(∂f(x1,x2)/∂x2) + ...
f(x1,x2,x3) = [x1 + x2]/[x3]²[x1 - x2] ∴ σ²y ≈ (∂f(x1,x2,x3)/∂x1)² σ²x1 + (∂f(x1,x2,x3)/∂x2)² σ²x2 +
(∂f(x1,x2,x3)/∂x3)² σ²x3 + 2σ²x1x2 (∂f(x1,x2,x3)/∂x1)(∂f(x1,x2,x3)/∂x2) + 2σ²x1x3 (∂f(x1,x2,x3)/∂x1)(∂f(x1,x2,x3)/∂x3) + 2σ²x2x3 (∂f(x1,x2,x3)/∂x2)(∂f(x1,x2,x3)/∂x3) ...
σ²y ≈ (∂f(x1, ..., xn)/∂x1)² σ²x1 + (∂f(x1, ..., xn)/∂x2)² σ²x2 + ...
+2σ²x1 x2 (∂f(x1, ..., xn)/∂x1)(∂f(x1, ..., xn)/∂x2) + ...

FIG. 20

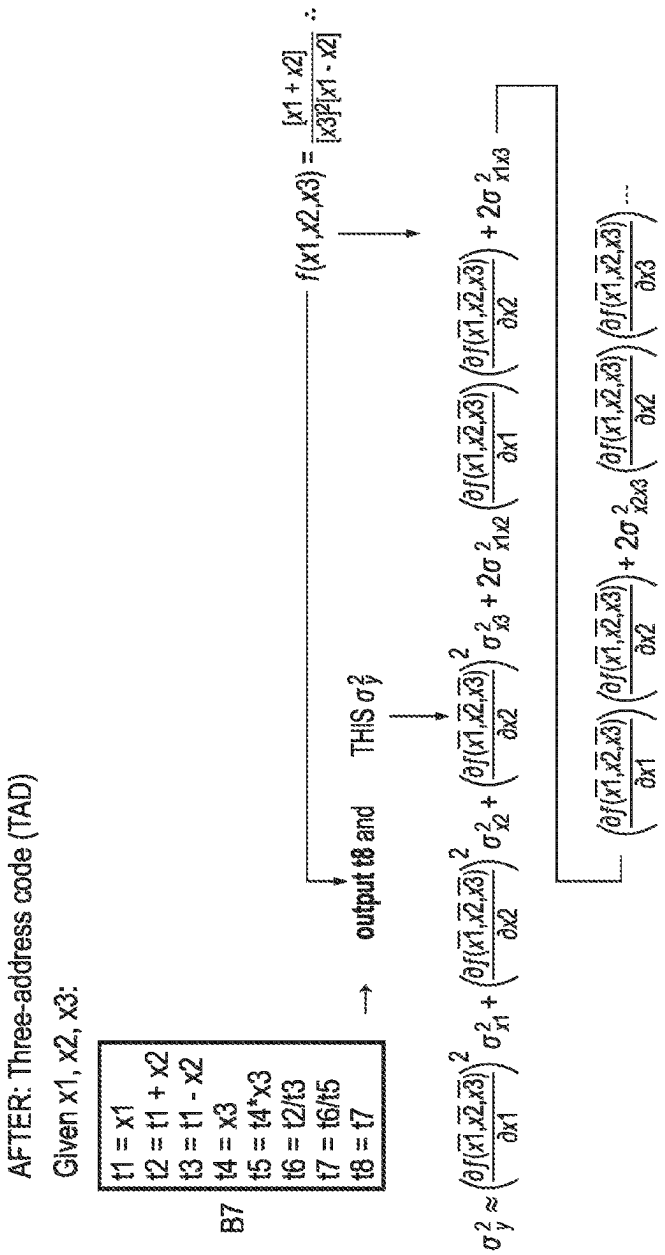

AFTER: Three-address code (TAD)
Given x1, x2, x3:
B7
t1 = x1
t2 = t1 + x2
t3 = t1 - x2
t4 = x3
t5 = t4*x3
t6 = t2/t3
t7 = t6/t5
t8 = t7
output t8 and
THIS σ_y^2
$f(x1,x2,x3) = \frac{[x1 + x2]}{[x3^2][x1 - x2]}$ ∴
$\sigma_y^2 \approx \left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x1}\right)^2 \sigma_{x1}^2 + \left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x2}\right)^2 \sigma_{x2}^2 + \left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x2}\right)^2 \sigma_{x3}^2 + 2\sigma_{x1x2}^2 \left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x1}\right)\left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x2}\right) + 2\sigma_{x1x3}^2$
$\left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x1}\right)\left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x2}\right) + 2\sigma_{x2x3}^2 \left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x2}\right)\left(\frac{\partial f(\bar{x1},\bar{x2},\bar{x3})}{\partial x3}\right) \cdots$

FIG. 21

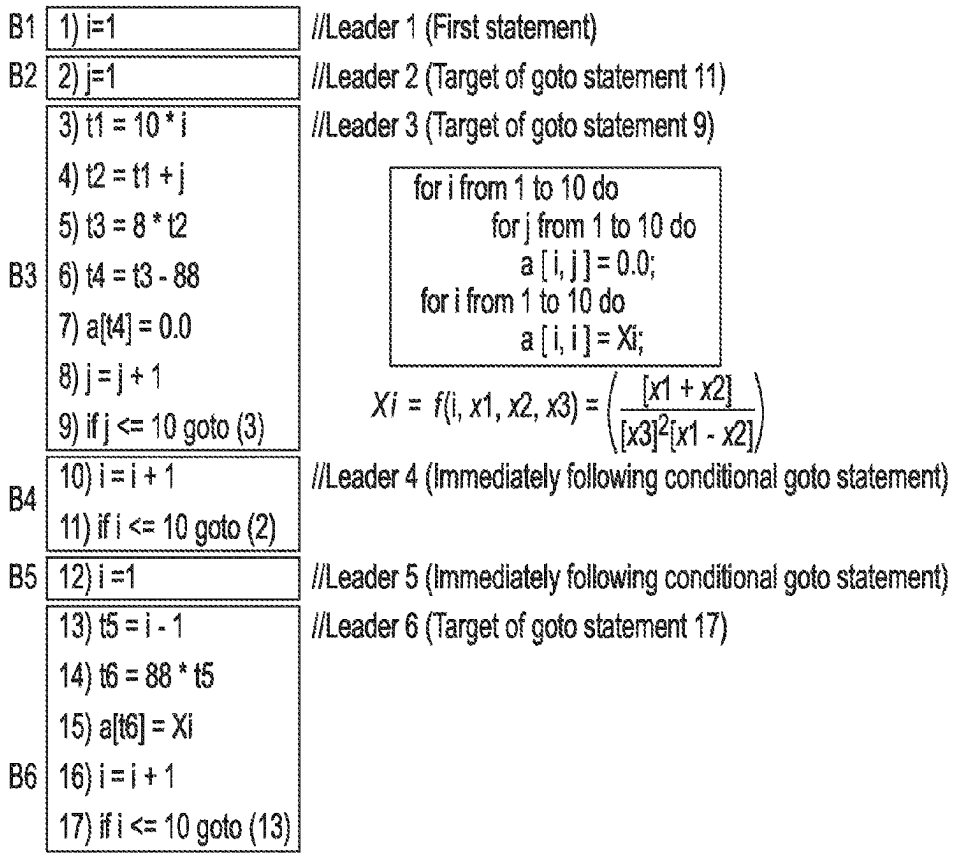

B1
1) i=1
//Leader 1 (First statement)
B2
2) j=1
//Leader 2 (Target of goto statement 11)
B3
3) t1 = 10 * i
4) t2 = t1 + j
5) t3 = 8 * t2
6) t4 = t3 - 88
7) a[t4] = 0.0
8) j = j + 1
9) if j <= 10 goto (3)
//Leader 3 (Target of goto statement 9)
for i from 1 to 10 do
for j from 1 to 10 do
a [i, j] = 0.0;
for i from 1 to 10 do
a [i, i] = Xi;
Xi = f(i, x1, x2, x3) = ([x1 + x2] / [x3]^2[x1 - x2])
B4
10) i = i + 1
11) if i <= 10 goto (2)
//Leader 4 (Immediately following conditional goto statement)
B5
12) i =1
//Leader 5 (Immediately following conditional goto statement)
B6
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = Xi
16) i = i + 1
17) if i <= 10 goto (13)
//Leader 6 (Target of goto statement 17)

FIG. 22

IMPROVEMENTS IN AND RELATING TO ENCODING AND COMPUTATION ON DISTRIBUTIONS OF DATA

[0001] This application claims priority from GB2107604.7 filed 27 May 2021 and from GB2107606.2 filed 27 May 2021, the contents and elements of which are herein incorporated by reference for all purposes.

FIELD OF THE INVENTION

[0002] The present invention relates to encoding, computation, storage and communication of distributions of data. The data may be distributions of data samples. The distributions of data may represent measurement uncertainty in measurement devices (e.g. sensors) and particularly, although not exclusively, may represent probability distributions.

BACKGROUND

[0003] Measurement apparatuses (e.g. sensors) pervade almost all aspects of modern life. From the monitoring of the operation of vehicles (e.g. engines and performance), to manufacturing apparatus and operations, power distribution networks, traffic control and telecommunications networks. The technical data produced by this monitoring is essential to helping manage these complex machines, structures and arrangements in a way that allows better efficiency and safety. The emergence of 'Big Data' analytics has grown hand-in-hand with the exponential growth in this technical data.

[0004] However, the ability to benefit from the analysis of such technical data sets is predicated on the accuracy and reliability of the data itself. If the data being analysed is of poor quality, then so too are the decisions that are made based on the results of that analysis. To quote and old adage: 'Garbage in; Garbage out'.

[0005] In all measurement apparatuses, no matter how sophisticated, the 'measurement' will never be identical to the 'measurand'. Following standard terminology from metrology, the true value of the input signal being measured by a measurement apparatus is known as the 'measurand'. Similarly, the estimate of the measurand obtained as the result of a measurement process, by a measurement apparatus, is known as the 'measurement'.

[0006] This difference, between the value of the 'measurand' and the value of the 'measurement', is either due to disturbances in the measurement instrument/sensor (e.g., circuit noise such as Johnson-Nyquist noise, or random telegraph noise, or transducer drift) or it is due to properties of the environment in which the measurement or sensing occurs (e.g. in LIDAR, so-called 'multipath' leading to anomalous readings).

[0007] The noise in the measurement instrument/sensor and the errors in measurements due to the environment or other non-instrument factors, can collectively be referred to as 'measurement uncertainty'.

[0008] Knowledge of measurement uncertainty associated with a technical data set permits the user to be informed about the uncertainty, and therefore the reliability, of the results of the analysis and decisions made on the basis of that data. Ignorance of this measurement uncertainty may lead to poor decisions. This can be safety-critical and is of paramount importance when the decision in question is being made by a machine (e.g. driverless car, an automated aircraft piloting system, an automated traffic light system etc.) unable to make acceptable risk-assessment judgements.

[0009] The present invention has been devised in light of the above considerations.

SUMMARY OF THE INVENTION

[0010] Uncertain data are ubiquitous. A common example is sensor measurements where the very nature of physical measurements means there is always some degree of uncertainty between the recorded value (the measurement) and the quantity being measured (the measurand). This form of measurement uncertainty is often quantified by performing repeated measurements with the measurand nominally fixed and observing the variation across measurements using statistical analysis. Such uncertainty in values, resulting from incomplete information on the values they should take, is of increasing relevance in modern computing systems. Modern computer architectures neither have support for efficiently representing uncertainty, let alone arithmetic and control-flow on such values. Computer architectures today represent uncertain values with single point values or "particle" values (i.e., data with no associated uncertainty distribution), usually by taking the mean value as the representation for use in computation. Hereafter, for conciseness, we refer to single point values (i.e., data with no associated distribution) as "particle" values.

[0011] The invention provides an encoding method for encoding information within a data structure for data relating to uncertainty real-world data (e.g., functional data associated with measurement values from a sensor, or values associated with the state of a physical system) for efficiently storing (e.g., physically in a register file, in a buffer memory or other memory storage) the information and/or efficiently propagating the stored information through subsequent computations.

[0012] The present invention may encode, represent and propagate distributional information/data (e.g., probability distributions, frequency distributions etc.) representing uncertainty in measurement data made by a measurement apparatus (e.g., sensor) or in values associated with the state of a physical system.

[0013] The technical considerations underlying the encoding method and the data structure it generates, relate to the intended use of the encoded data: namely, in computations performed on distributions representing uncertainty in data values. These considerations allow a computing architecture to work efficiently using parameters within the data structure that encode a probability distribution describing uncertainty in real-world data. The result is to provide an efficient method for generating new parameters which are consistent/comply with a common data structure format and requirements, and which encode a probability distribution representing the result of applying an arithmetic operation on two (or more) other probability distributions. This allows further such computations to be applied to the new parameters in a consistent manner when calculating uncertainty in quantities calculated by applying further arithmetic operations. As a result, computations performed upon two or more "particle" values to obtain a new "particle" value may also be concurrently performed on the corresponding uncertainty distributions of the two or more "particle" values so as to obtain a new uncertainty distribution associated with the new "particle" value efficiently and in a consistent manner.

[0014] In a first aspect, the invention may provide a method computer-implemented method for the encoding of, and computation on, distributions of data, the method comprising:

[0015] obtaining a first set of data items;

[0016] obtaining a second set of data items;

[0017] generating a first tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the first set;

[0018] generating a second tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the second set in which the parameters used to encode the distribution of the data items of the second set are the same as the parameters used to encode the distribution of the data items of the first set;

[0019] generating a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding a probability distribution representing the result of applying an arithmetic operation on the first probability distribution and the second probability distribution;

[0020] outputting the third tuple.

[0021] In this way, each tuple provides a data structure within which distributional information is encoded that represents a probability distribution (e.g., uncertainty in an associated "particle" value). A reference herein to a "tuple" may be considered to include a reference to a data structure consisting of multiple parts defining an ordered set of data constituting a record, as is commonly understood in the art. In mathematics, a tuple is a finite ordered list (sequence) of elements such as a sequence (or ordered list) of n elements, where n is a non-negative integer. The parameters contained within the first, second and third tuples according to preferred examples of the invention are, therefore, ordered according to a common parameter ordering or sequence which controls/instructs a computing system implementing the method on how to calculate a new distributional data (i.e., a new tuple, consistently encoded) to be associated with a new "particle" data item generated by doing arithmetic operations on two or more other "particle" data items each having their own respective associated distributional data (i.e., a respective tuple, consistently encoded). These encoded data structures (tuples) are distinct from the data (distributions) itself. The data structure may be automatically recognised by the computing system implementing the method, such that it may operate accordingly. In this way, a causal link exists between the encoding underlying the data structure, the interpretation of the data structure by a computing system for performing calculations using the data within the data structure, and the technical/operational efficiencies gained by the computing system when calculating new distributional information for a calculated new "particle" data item.

[0022] A "variable" may be considered to be a symbol which works as a placeholder for expression of quantities that may vary or change. For example, a "variable" may be used to represent the argument of a function or an arbitrary element of a set. A parameter, in mathematics, may be considered to be a variable for which the range of possible values identifies a collection of distinct cases in a problem. For example, any equation expressed in terms of parameters is a parametric equation. For example, the general equation of a straight line in gradient-intercept form, $y=mx+c$, in which m and c are parameters, is an example of a parametric equation. Different instances of this equation may be said to use "the same" parameters (i.e. a gradient parameter, m, and an intercept parameter, c) whether or not the actual values of those parameters are the same: first instance, $y=m_1x+c_1$; second instance, $y=m_2x+c_2$ use the same parameters (gradient-intercept) whether or not the actual values of m_1 and m_2 are equal (i.e. $m_1=m_2$) and whether or not the actual values of c_1 and c_2 are equal (i.e. $c_1=c_2$). In this sense, any two tuples containing parameters encoding a respective probability distribution may be said to use "the same" parameters (e.g. have the same use of parameters). As a non-limiting example, discussed in more detail below, the parameters used to encode a distribution of data items may be the parameters of "position", x_i, of Dirac-δ functions and "probability", p_i. Different distributions, for different data sets, may use these two parameters and therefore use "the same" parameters as each other. Of course, the actual values assigned to these parameters (position, probability) for the data in each set, as defined in this shared parametric form, are generally not the same. The use of the same parameters permits the method to achieve the same format of representation using the tuples to encode the distribution of the data items, and from that format, the same format of representation of the distributions themselves may be reproduced.

[0023] Preferably, the outputting of the third tuple comprises one or more of: storing the third tuple in a memory (e.g., in a register file, in a buffer memory or other memory storage); transmitting a signal conveying the third tuple (e.g., via an electrical signal or via an electromagnetic signal/carrier-wave). Accordingly, the encoding method improves the efficiency of storage and/or transmission of the probability distribution information encoded by the data structure of the third tuple.

[0024] According to the method, the arithmetic operation may comprise one or more of: addition; subtraction; multiplication; division, or more complex arithmetic operations e.g., fused multiplication and addition or square root, or any bivariate operation, e.g., exponentiation, by expressing them in terms of the aforementioned basic arithmetic operations, as would be readily apparent to the skilled person. Thus, just as new "particle" data item may be generated by doing any such arithmetic operations on two or more other "particle" data items each having their own respective associated distributional data, similarly, new distributional information (e.g., the third tuple) may be calculated for the new "particle" data item by applying the same arithmetic operations on the distributional data associated with the two or more other "particle" data items (e.g., the first and second tuples).

[0025] The third tuple may contain parameters encoding a probability distribution characterising the distribution of the data items of a third set of data items in which the parameters used to encode the distribution of the data items of the third set are the same as the parameters used to encode the distribution of the data items of the first set. Accordingly, third tuple may be a data structure containing parameters ordered according to a parameter ordering or sequence which is in common with the parameter ordering or sequence employed in the first tuple. This has the advantage of providing a common data structure in the first, second and third tuples when anyone or more of these tuples is subsequently used for controlling/instructing a computing system to calculate a further new distributional data (i.e., a new

tuple, consistently encoded) to be associated with a further new "particle" data item generated by doing arithmetic operations on two or more "particle" data items each having their own respective associated distributional data (i.e., a respective tuple, consistently encoded).

[0026] Preferably, the first set of data comprises samples of a first random variable, and the second set of data comprises samples of a second random variable.

[0027] Preferably, the method comprises outputting the first tuple by one or more of: storing the first tuple in a memory (e.g., in a register file, in a buffer memory or other memory storage); transmitting a signal conveying the first tuple (e.g., via an electrical signal or via an electromagnetic signal/carrier-wave). Preferably, the method comprises outputting the second tuple by one of more of: storing the second tuple in a memory (e.g., in a register file, in a buffer memory or other memory storage); transmitting a signal conveying the second tuple (e.g., via an electrical signal or via an electromagnetic signal/carrier-wave).

[0028] The method may comprise obtaining the output first tuple by one or more of: retrieving the output first tuple from a memory; receiving a signal conveying the output first tuple. The method may comprise obtaining of the output second tuple by one or more of: retrieving the output second tuple from a memory; receiving a signal conveying the output second tuple. The method may comprise generating the third tuple using parameters contained within the obtained first tuple and within the obtained second tuple.

[0029] Desirably, the first tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the first set. For example, the position of data items may be positions of Dirac delta functions. Preferably, the second tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the second set. For example, the position of data items may be positions of Dirac delta functions. Desirably, the third tuple contains parameters encoding the position of data items within a probability distribution characterising the distribution of the data items of a third set of data items. For example, the position of data items may be positions of Dirac delta functions.

[0030] Desirably, the first tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the first set. Preferably, the second tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the second set. Desirably, the third tuple contains parameters encoding the position and/or width of data intervals within a probability distribution characterising the distribution of the data items of a third set of data items.

[0031] Desirably, the first tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the first set. For example, the probability of a data item encoded within the first tuple may be an amplitude or a weighting of a Dirac delta function, which may be positioned according to one or more parameters encoding the position of data item. Preferably, the second tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the second set. For example, the probability of a data item encoded within the second tuple may be an amplitude or a weighting of a Dirac delta function, which may be positioned according to one or more parameters encoding the position of data item. Desirably, the third tuple contains parameters encoding the probability of data items within a probability distribution characterising the distribution of the data items of a third set of data items. For example, the probability of a data item encoded within the third tuple may be an amplitude or a weighting of a Dirac delta function, which may be positioned according to one or more parameters encoding the position of data item.

[0032] Desirably, the first tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the first set. Preferably, the second tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the second set. Desirably, the third tuple contains parameters encoding the value of one or more statistical moments of a probability distribution characterising the distribution of the data items of a third set of data items.

[0033] Desirably, the probability distribution characterising the distribution of the data items of the first set comprises a distribution of Dirac delta functions. Preferably, the probability distribution characterising the distribution of the data items of the second set comprises a distribution of Dirac delta functions. Desirably, the probability distribution characterising the distribution of the data items of the third set comprises a distribution of Dirac delta functions.

[0034] Desirably, the first tuple is an N-tuple in which N>1 is an integer. Preferably the second tuple is an N-tuple in which N>1 is an integer. Desirably, the third tuple is an M-tuple for which $N^2/2<M<2N^2$ in which N>1 is an integer. For example, for SoDD-based representations discussed in more detail below (i.e., all except CMR), if N is the memory usage of a given representation method with N_{dd} Dirac deltas, the initial M-tuple calculated as arithmetic propagation of the input N-tuples will encode N_{dd}^2 Dirac deltas using $2N_{dd}^2$ numbers (except for PQHR for which N_{dd}^2 numbers suffice). In other words, (except for PQHR) $M=2N_{dd}^2$. Table 1, below gives the relationship between N and N_{dd} for a given method. For TTR and MQHR $N=2N_{dd}$, which implies $M=N_{dd}^2/2$.

[0035] The method may comprise reducing the size of the third tuple (M-tuple representation) to be the same size (N-tuple) as the first tuple and the second tuple, to provide a reduced third tuple which is an N-tuple. This has the advantage of enabling a fixed size for the tuples that one calculates with (i.e. all being N-tuples), to enable further calculations with the derived tuples (i.e. the results of the arithmetic on distributions). This may be achieved by considering the data represented by the third tuple as being a new "obtained" data set (i.e. an obtained third set of data items, obtained as the result of the arithmetic operation) and therefrom generating a "compacted" third tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the third set. The parameters used in the compacted third tuple to encode the distribution of the data items of the third set, may not only be the same as the parameters used to encode the distribution of the data items of the first set, but also the compacted third tuple may be constructed to be the same size (i.e. an N-tuple; the size is reduced from M to N) as both the first and second

tuples. In this way, any of the methods and apparatus disclosed herein for use in generating a tuple from an obtained set of data items, may equally be applied to the output result of applying the arithmetic operation on distributions to allow that output to be represented as a tuple of the same size as the tuples representing the data set to which the arithmetic was applied. References herein to the "third tuple" may be considered to include a reference to a "compacted third tuple", as appropriate.

[0036] In another aspect, the invention may provide a computer program product comprising a computer program which, when executed on a computer, implements the method according to the invention described above, in its first aspect.

[0037] In a second aspect, the invention may provide an apparatus for implementing the encoding of, and computation on, distributions of data, the apparatus comprising:

[0038] a memory for storing a first set of data items and a second set of data items;

[0039] a processor configured to perform the following processing steps:

[0040] generate a first tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the first set;

[0041] generate a second tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the second set in which the parameters used to encode the distribution of the data items of the second set are the same as the parameters used to encode the distribution of the data items of the first set;

[0042] generate a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding a probability distribution representing the result of applying an arithmetic operation on the first probability distribution and the second probability distribution; and,

[0043] output the third tuple.

[0044] Preferably, the processor may be implemented as a microprocessor, or a dedicated digital logic circuit, or an analogue circuit configured to perform the processing steps. Preferably, the apparatus is configured to output the third tuple by one or more of: storing the third tuple in a memory; transmitting a signal conveying the third tuple.

[0045] Preferably, the apparatus is configured to output the first tuple by one or more of: storing the first tuple in a memory; transmitting a signal conveying the first tuple. Preferably, the apparatus is configured to output the second tuple by one or more of: storing the second tuple in a memory; transmitting a signal conveying the second tuple.

[0046] The apparatus is configured to obtain the output first tuple by one or more of: retrieving the output first tuple from a memory; receiving a signal conveying the output first tuple. The apparatus is configured to obtain the output second tuple by one or more of: retrieving the output second tuple from a memory; receiving a signal conveying the output second tuple. The apparatus is configured to generate the third tuple using parameters contained within the obtained first tuple and within the obtained second tuple.

[0047] The apparatus may be configured to perform the step of outputting of the first tuple by one or more of: storing the first tuple in a memory; transmitting a signal conveying the first tuple. The apparatus may be configured to perform the step of outputting the second tuple by one or more of: storing the second tuple in a memory; transmitting a signal conveying the second tuple. The apparatus may be configured to perform the step of outputting the third tuple by one or more of: storing the third tuple in a memory; transmitting a signal conveying the third tuple.

[0048] The apparatus may be configured to perform the step of obtaining of the output first tuple by one or more of: retrieving the output first tuple from a memory; receiving a signal conveying the output first tuple. The apparatus may be configured to perform the step of obtaining of the output second tuple by one or more of: retrieving the output second tuple from a memory; receiving a signal conveying the output second tuple.

[0049] The apparatus may be configured to perform the arithmetic operation comprising one or more of: addition; subtraction; multiplication; division.

[0050] The apparatus may be configured such that the third tuple contains parameters encoding a probability distribution characterising the distribution of the data items of a third set of data items in which the parameters used to encode the distribution of the data items of the third set are the same as the parameters used to encode the distribution of the data items of the first set.

[0051] The apparatus may be configured such that the first tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the first set. The apparatus may be configured such that the second tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the second set. The apparatus may be configured such that the third tuple contains parameters encoding the position of data items within a probability distribution characterising the distribution of the data items of a third set of data items.

[0052] The apparatus may be configured such that the first tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the first set. The apparatus may be configured such that the second tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the second set. The apparatus may be configured such that the third tuple contains parameters encoding the position and/or width of data intervals within a probability distribution characterising the distribution of the data items of a third set of data items.

[0053] The apparatus may be configured such that the first tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the first set. The apparatus may be configured such that the second tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the second set. The apparatus may be configured such that the third tuple contains parameters encoding the probability of data items within a probability distribution characterising the distribution of the data items of a third set of data items.

[0054] The apparatus may be configured such that the first tuple contains parameters encoding the value of one or more statistical moments of the probability distribution character-

ising the distribution of the data items of the first set. The apparatus may be configured such that the second tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the second set. The apparatus may be configured such that the third tuple contains parameters encoding the value of one or more statistical moments of a probability distribution characterising the distribution of the data items of a third set of data items.

[0055] The apparatus may be configured such that the probability distribution characterising the distribution of the data items of the first set comprises a distribution of Dirac delta functions. The apparatus may be configured such that the probability distribution characterising the distribution of the data items of the second set comprises a distribution of Dirac delta functions. The apparatus may be configured such that the probability distribution characterising the distribution of the data items of the third set comprises a distribution of Dirac delta functions.

[0056] The apparatus may be configured such that the first tuple is an N-tuple in which N>1 is an integer. The apparatus may be configured such that the second tuple is an N-tuple in which N>1 is an integer. The apparatus may be configured such that the third tuple is an M-tuple for which $N^2/2<M<2N^2$ in which N>1 is an integer.

[0057] In another aspect, the invention may provide a computer programmed with a computer program which, when executed on the computer, implements the method according described above, in the first aspect of the invention.

[0058] The invention includes the combination of the aspects and preferred features described except where such a combination is clearly impermissible or expressly avoided. This means that, for example, the invention in its first aspect (method) may be implemented according to the invention in its third aspect (below) by providing a microarchitecture to implement the method. Similarly, the invention in its second aspect (apparatus) may be implemented according to the invention in its fourth aspect (below) as a microarchitecture.

[0059] Computers carry out arithmetic with point-valued numbers. The data that dominate contemporary computing systems are however from measurement processes such as sensors. All measurements are inherently uncertain and this uncertainty is often characterized statistically and forms aleatoric uncertainty. In addition, many other contemporary applications of probability distributions, such as machine learning, comprise models which also have inherent epistemic uncertainty (e.g., on the weights of a neural network). Hardware and software can exploit this uncertainty in measurements for improved performance as well as for trading performance for power dissipation or quality of results. All of these potential applications however stand to benefit from more effective methods for representing arbitrary real-world probability distributions and for propagating those distributions through arithmetic.

[0060] The inventors have realised that each real number in the domain of a distribution may be described by a Dirac delta with some probability mass located at the value of the real number. This representation of a distribution in terms of Dirac deltas within its domain is different from that of a probability mass function (PMF), where the domain of the distribution is by definition discrete-valued and integrals can be replaced with sums. Because the Dirac delta distribution is not a function but rather is a distribution, the representation of distributions as a collection of Dirac deltas is also different from the concept of a probability density function (PDF). The inventors therefore refer, herein, to the representation comprising a sum of Dirac deltas as a probability density distribution (PDD).

Probability Density Distributions (PDDs) and Finite-Dimensional Representations

[0061] In general, a real-valued random variable is characterized by its PDD defined on the real numbers.

[0062] Because we are concerned with computer representations, the information on such a probability distribution is represented by finitely many real number representations. As real number representations of finite size (e.g., 32-bit floating-point representation) provide a discrete and finite representation of the real line, it is conventional to work with probability mass functions (PMFs) instead of PDDs. However, in the present disclosure we ignore the error in representation of real numbers and assume that each real number can be represented exactly. This removes the discrete nature of all permissible values taken by a random variable and as a consequence we employ a formalism that uses PDDs instead of PMFs. Thus, one may then encapsulate the finite capabilities of a computer in representing a given abstract PDD by representing it using finitely-many, say N, (exactly-represented) real numbers, to which we will refer as a finite-dimensional representation of size N.

[0063] Computation on PDDs requires, first, an algorithm for calculating the finite-dimensional representation from a given description of the PDD of a random variable and, second, an algorithm for propagating such representations under given arithmetic operations. In the present disclosure, we refer to these two types of algorithms as finite-dimensional representation methods and arithmetic propagation methods, respectively. The present disclosure presents examples of such methods for computation with random variables.

[0064] The description of the PDD can be an analytical expression or it can be in the form of samples drawn from a distribution which itself gives rise to a discrete PDD. The present disclosure presents methods that are generic to calculation of representations from any PDD, whether continuous or discrete. The present disclosure is relevant in its application to computations in aleatoric uncertainty in physical measurements (e.g., noisy sensors in autonomous vehicles) and epistemic uncertainty (e.g., weights in a neural network). The present disclosure is relevant in its application to representations calculated from samples drawn from a discrete distribution. The present disclosure considers arithmetic propagations in the case of mutually independent random variables.

Real-Valued Random Variables as Generalizations of Real Numbers.

[0065] In the generalized context of real-valued random variables, a given real number $x_0 \in \mathbb{R}$ can be thought of as a random variable whose PDD is concentrated at a single point, namely at $x=x_0$. The PDD of such a point-valued variable is given by the Dirac delta distribution. Let $\mathcal{C}(\mathbb{R})$ denote the space of continuous functions on $\mathbb{R}$. We will refer to the bounded linear functional $\delta: \mathcal{C}(\mathbb{R}) \rightarrow \mathbb{R}$ defined by:

$$\delta(f) = \int_{\mathbb{R}} f(x)\delta(x)dx := f(0), \forall\, f \in C(\mathbb{R})$$

as the Dirac delta distribution. Note that 8 is not a function as the conventional integral notation may misleadingly infer. Yet, we will still employ the notation $x_0 := \delta(x - x_0)$ interchangeably to stand for a Dirac delta distribution with evaluation at $x = x_0$, as this notation is in line with application of change of variables in the integral notation. Henceforth, we will regard all real numbers $x_0 \in \mathbb{R}$ as point-valued random variables with the associated PDD given by δ_{x0}.

[0066] Herein we refer to a PDD f_x is a 'sum of Dirac deltas' (SoDD) representation when it is of the form:

$$f_X(x) = \sum_{n=1}^{N} p_n \delta(x - x_n)$$

[0067] Where $p_n \in [0,1]$ with:

$$\sum_{n=1}^{N} p_n = 1$$

Finite-Dimensional Representations

[0068] To provide some examples, for a better understanding of the invention, the present disclosure introduces five finite-dimensional representation methods. Algorithms are presented to calculate the representation for a given method from a given PDD, as well as algorithms to derive an approximating PDD from a given finite-dimensional representation together with rules governing their propagation under arithmetic operations of addition and multiplication.

Finite-Dimensional Representation Methods

[0069] For a given finite-dimensional representation method, $\mathcal{M}$, a representation $\mathcal{R}^N_{\mathcal{M}}$ is a mapping of random variables (or their PDDs) into $\mathbb{R}^N$, where we call N the dimension of the representation. This disclosure describes for each method $\mathcal{M}$ considered here the corresponding representation $\mathcal{R}^N_{\mathcal{M}}$. The information on f_x may be provided by data samples.

[0070] For a given SoDD-based representation we denote by N_{dd} the number of Dirac deltas in the corresponding SoDD distribution. N_{dd} is not same as the dimension N, which is equal to the minimum number of real numbers required to specify the approximating SoDD distribution within the context of given representation method. In general, we have $N_{dd} < N$. Table 1 summarizes the relations of N_{dd} to N for each examples of SoDD-based representation methods disclosed herein and discussed in more detail below.

TABLE 1

Representation	N_{dd}	N_k	N_{dd} for N = 16
RQHR	N − 2	$N_{dd}^k + 2k$	14
PQHR	N/2	N_{dd}^k	16
MQHR	N/2	$2N_{dd}^k$	8
TTR	N/2	$2N_{dd}^k$	8

Example 1: Regularly-Quantized Histogram Representation (RQHR)

[0071] This representation method assumes that the value range R(X) of a given variable X is known and finite, i.e., $|R(X)| < 1$. Note that this is always the case if the PDD of X is constructed from finitely many samples. The underlying idea is to create an approximating SoDD distribution by dividing R(X) into N_{dd} equal-sized (regular) intervals and to represent the given PDD by a weighted Dirac delta located at the center of each interval with the weight equal to the probability mass of the PDD over the corresponding interval. If we define $x_0 := \inf R(X)$ and $t_x := |R(X)|/N$, then the aforementioned regular intervals have the form:

$$I_n = (x_0 + (n-1)l_X, x_0 + nl_X),\ 1 \le n \le N_{dd}$$

[0072] The probabilities p_n over these intervals are given by:

$$p_n = \int_{I_n} f_X\, dx,\ 1 \le n \le N_{dd}$$

[0073] Then, the N-dimensional regularly-quantized histogram representation (RQHR) of X is defined as the ordered N-tuple:

$$\mathcal{R}^N_{RQHR}(X) := (x_0, l_X, p_1, \ldots, p_{N_{dd}})$$

[0074] Here $N_{dd} = N - 2$. The corresponding approximating SoDD distribution for the RQHR in has the form:

$$\tilde{f}_X^{RQHR}(x) := \sum_{n=1}^{N_{dd}} p_n \delta\left(x - \left(x_0 + \left(n - \frac{1}{2}\right)l_X\right)\right)$$

Example 2: Probability-Quantized Histogram Representation (PQHR)

[0075] As in the RQHR method, where the value range of X was regularly quantized, this method also quantizes the range into intervals, but with intervals having the same probability mass.

[0076] First, define the interval:

$$I_n = F_X^{-1}\left(\left[\frac{n-1}{N_{dd}}, \frac{n}{N_{dd}}\right)\right),\ 1 \le n \le N_{dd}$$

[0077] The function F_X^{-1} is the inverse of F_X, where F_X is the cumulative distribution function of the random variable X.

[0078] The expected value μ_{Xn} of X_n within a given interval, of probability mass common to all intervals, is:

$$\mu_{X_n} = \left(\int_{I_n} f_X(x)\,dx\right)^{-1} \int_{I_n} x f_X(x)\,dx = N_{dd} \int_{I_n} x f_X(x)\,dx$$

[0079] Then, the N-dimensional probability-quantized histogram representation (PQHR) of X is defined as the ordered N-tuple:

$$\mathcal{R}^N_{PQHR}(X) := \left(x_{X_1}, \ldots, \mu_{X_{N_{dd}}}\right)$$

[0080] Here N_{dd}=N. The corresponding approximating SoDD distribution for the PQHR in has the form:

$$\tilde{f}_X^{PQHR}(x) := \frac{1}{N_{dd}} \sum_{n=1}^{N_{dd}} \delta_{\mu X_n}$$

Example 3: Moment-Quantized Histogram Representation (MQHR)

[0081] Given a random variable X with PDD f_X and expected values μ_X, let CMF_X be the cumulative moment function of X, defined by:

$$CMF_X(x) = \int_{-\infty}^{x} |\xi - \mu_X| f_X(\xi)\,d\xi$$

[0082] Note that CMF_X is monotone increasing and thus we have:

$$\lim_{x \to +\infty} CMF_X(x) = TM_X$$

[0083] where TM_X is the total moment of X around μ_X (i.e. setting x_0=μ_X) and given by:

$$TM_X(x_0) := \int_{\mathbb{R}} |x - x_0| f_X(x)\,dx.$$

[0084] Here we assume that TM_X is finite, which is true for the practical cases of finitely-many sample generated PDDs. We partition the value range of X, e.g., the real line, by quantizing the cumulative moment TM_X. Thus, we define the intervals:

$$I_n := CMF_X^{-1}\left(\left(\frac{(n-1)TM_X}{N_{dd}}, \frac{nTM_X}{N_{dd}}\right)\right),$$
$$1 \le n \le N_{dd}$$

[0085] As before, consider the expected value of X given $X \in I_n$:

$$x_n := E[X | X \in I_n] = \left(\int_{I_n} f_X(x)dx\right)^{-1} \int_{I_n} x f_X(x)dx = \frac{1}{p_n} \int_{I_n} x f_X(x)dx,$$

Here,

$$p_n := \int_{I_n} f_X(x)dx,\ 1 \le n \le N_{dd}$$

[0086] Then, the N-dimensional moment-quantized histogram representation (MQHR) of X is defined as the ordered N-tuple:

$$\mathcal{R}^N_{MQHR}(X) := \left(x_1, \ldots, x_{N_{dd}}, p_1, \ldots, p_{N_{dd}}\right)$$

[0087] Here, N_{dd}=N/2. The corresponding approximating SoDD distribution for the MQHR has the form:

$$\tilde{f}_X^{MQHR}(x) := \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n)$$

Example 4: Telescoping Torques Representation (TTR)

[0088] For a random variable X with PDD f_X and expected value μ_X we define:

$$\Omega_- := \{X \in \mathbb{R} : x < \mu_X\},\ \Omega_+ := \{X \in \mathbb{R} : x \ge \mu_X\}$$

and

$$X_- := X | X \in \Omega_-,\ X_+ := X | X \in \Omega_+$$

[0089] The following algorithm underlies the TTR method.

[0090] Let X be a random variable with PDD f_X and expected value μ_X. There is a unique approximating PDD $\tilde{f}_X$ in the form of a SoDD distribution of size 2, that is:

$$\tilde{f}_X(x) = p_-\delta(x - x_-) + p_+\delta(x - x_+), \text{ with } p_- + p_+ = 1$$

such that:

$$\int_{\mathbb{R}} x \tilde{f}_X(x)dx = \mu_X, \tag{1}$$

$$\int_{\mathbb{R}} |x - \mu_X| \tilde{f}_X(x)dx = \int_{\mathbb{R}} |x - \mu_X| f_X(x)dx, \tag{2}$$

$$\int_{\Omega_+} \tilde{f}_X(x)dx = Pr_X(\Omega_+). \tag{3}$$

[0091] Moreover, the above parameters $p_{\pm}$ and $x_{\pm}$ of this unique SoDD distribution of size 2 satisfy:

$$p_- = Pr_X(\Omega_-),\ p_+ = Pr_X(\Omega_+),\ x_- = E[X_-],\ x_+ = E[X_+]$$

[0092] This circulates around the intuition that the expected value is the centre of mass of the PDD f_X, as in a rod with mass density described by f_X would stay in balance if supported from $x=\mu_X$, and one has the full knowledge of counterbalancing torques (the 2 Dirac deltas in SoDD distribution), i.e., forces (weights or probability masses) and distances (positions), provided that one knows the magnitude of total moment applied from both sides (property (2), above) and the ratio of weights on both sides (property (3), above). One can iterate the idea of this unique 2-SoDD distribution that holds the torque information around the expected value in a telescoping manner, which gives rise to the name of TTR method, to arrive at a unique 2^n-SoDD distribution that contains all the torque information around expected values of X restricted to special inductively-defined intervals of the form Ω_- and Ω_+ given above.

[0093] The TTR of X is formulated inductively as follows. We define the 0^{th}-order TTR of X to be its expected value, recorded as the $N=2\times2^0=2$-tuple:

$$\mathcal{R}^2_{TT}(X) = (x_1,\ p_1) = (\mu_X,\ 1)$$

[0094] This tuple represents a real number as taking the value μ_X with probability 1, with the corresponding approximating SoDD distribution of size 1:

$$\tilde{f}^{TTR}_X(x) = \delta(x-\mu_X)$$

[0095] The 1^{st}-order TTR of X is given by the $N=2\times2^1=4$-tuple:

$$\mathcal{R}^4_{TT}(X) = (x_1,\ x_2,\ p_1,\ p_2) = (x_-,\ x_+,\ p_-,\ p_+)$$

[0096] This corresponds to the unique SoDD distribution of size 2 given by the equations above. In general, n^{th}-order TTR of X will be a $N=2\times2^n$-tuple:

$$\mathcal{R}^{2^{n+1}}_{TT}(X) = (x_1,\ \ldots\ ,\ x_{2^n},\ p_1,\ \ldots\ ,\ p_{2^n})$$

[0097] This has a corresponding SoDD distribution of size $N_{dd}=2^n$. For higher order TTRs we will introduce inductive extensions of definitions given above. For this we introduce the following indexing notation. Let α stand for a concatenated sequence of "−" and "+" signs, e.g., $\alpha=-+-$ or $\alpha=+$, let $|\alpha|$ be the length of the sequence α and let $\alpha=+$ and $\alpha=-$ mean the sequences of length $|\alpha|+1$ that one obtains by concatenating a "−" sign or a "+" sign to the right end of α, respectively. We then inductively define:

$$\Omega_{\alpha-} := \{x \in \Omega_\alpha : x < E[X_\alpha]\},\ \Omega_{\alpha+} := \{x \in \Omega_\alpha : x \geq E[X_\alpha]\}$$

and

$$X_{\alpha-} := X|X \in \Omega_{\alpha-},\ X_{\alpha+} := X|X \in \Omega_{\alpha+}$$

[0098] Note that for a given $n\geq1$ there are 2^n distinct sequences of length n and corresponding 2^n domains Ω_α which are indeed intervals. These 2^n intervals partition R and they are ordered in accordance with the enumeration φ_n where, for $1\leq i\leq2^n$, $\varphi_n(i)$ is the sequence obtained by replacing 0's and 1's in the length-n binary representation of i−1 by "−" and "+" signs, respectively. For instance, $\varphi_3(1)=---$ and $\varphi_4(5)=-+--$. Then, the n^{th}-order TTR of X is defined as the ordered N-tuple ($N=2^{n+1}$):

$$\mathcal{R}^N_{TTR}(X) := \left(\mu_{X_{\varphi_n(1)}},\ \ldots\ ,\ \mu_{X_{\varphi_n(N_{dd})}},\ p_{\varphi_n(1)},\ \ldots\ ,\ p_{\varphi_n(N_{dd})}\right)$$

Here:

$$p_{\varphi_n(i)} := Pr_X\left(\Omega_{\varphi_n(1)}\right)$$

and

$$N_{dd} = \frac{N}{2} = 2^n$$

[0099] The corresponding approximating SoDD distribution for the TTR has the form:

$$\tilde{f}^{TTR}_X(x) := \sum_{i=1}^{N_{dd}} p_{\varphi_n(i)}\delta\left(x-\mu_{X_{\varphi_n(n)}}\right)$$

Example 5: Centralized Moments Representation (CMR)

[0100] The n^{th} centralized moment of X, $\mu_n(X)$, where $n\geq0$, is defined as:

$$\mu_0(X) := 1,$$

$$\mu_X := \mu_1(X) := \int_{\mathcal{R}} x f_X(x)dx,$$

$$\mu_n(X) := \int_{\mathcal{R}} (x-\mu_X)^n f_X(x)dx,$$

$$n \geq 2.$$

[0101] Then, the N-dimensional centralized moment representation (CMR) of X is defined as the ordered N-tuple

$$\mathcal{R}^N_{CMR}(X) := (\mu_1(X),\ \ldots\ ,\ \mu_N(X))$$

[0102] This definition of centralized moments differs from the typical definition in the literature: The first centralized moment, $\mu_1(X)$, is conventionally calculated centred at the expected value and is thus 0. By contrast, in the present

disclosure we define $\mu_1(X)$ to equal the expected value of X. This formulation allows us to define all the centralized moments recursively in terms of lower-order centralized moments, with the zeroth-order centralized moment $\mu_0(X)$: =1 as the base case of the recursion.

[0103] Because the PDD is a collection of Dirac delta distributions, there are several methods of approximating PDD from the knowledge of centralized moments. Possible methods include Gram-Charlier series [ref 1] and Edgeworth-type expansions [ref 2]. In addition, one may use the formal Edgeworth series as described by Bhattacharya et al. [ref 3] to construct approximating PDDs from CMRs.

Arithmetic Operations

[0104] In this part we outline the rules of arithmetic operations (propagations) of finite-dimensional representations we have just introduced. For a given two random variables X and Y the following disclosure describes how one propagates representations of X and Y of given type to find the same type of representation for the random variable Z:=(X; Y). The propagation rules for all SoDD-based representation methods, i.e., all except CMR, are the same, as they apply to the underlying SoDD distribution rather than the method of representation. Consequently, the following disclosure describes two sets of arithmetic propagation rules, one for CMR and one for SoDD-based representations. For sake of simplicity, in the case of arithmetic propagation of CMR we will restrict our attention to the arithmetic operations of addition and multiplication only. However, it is to be understood that an addition or multiplication of a value to a variable in a SoDD-based representation results in an offset or scaling of the position of the Dirac deltas of the representation and, therefore, the subtraction of two uncertain variables may be achieved by negating the SoDD-based representation of the subtrahend (i.e. the SoDD-based representation of the quantity or number to be subtracted from another) using multiplication with −1. For the case of division of variables in a SoDD-based representation, we define it as a multiplication using the reciprocal of the divisor variable. The reciprocal of a variable in a SoDD-based representation can be constructed by calculating the reciprocals of the input Dirac deltas positions. For example, division of two random variables of a SoDD Dirac mixture representation may be implemented as a multiplication of the dividend with the reciprocal of the divisor. The latter may be constructed by calculating the reciprocals of the positions of its Dirac mixture representation.

Example 5: Propagation of SoDD-based Representations

[0105] Suppose we are given an N-dimensional SoDD-based representation of fixed type, i.e., RQHR, PQHR, MQHR or TTR, and let us call it FDR. For two mutually independent random variables X and Y with approximating SoDD distributions of size N_{dd} given as:

$$\tilde{f}_X^{FDR}(x) = \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n) \text{ and}$$

-continued

$$\tilde{f}_Y^{FDR}(y) = \sum_{n=1}^{N_{dd}} q_n \delta(y - y_n).$$

[0106] Propagation of these two PDDs in the form of SoDD distributions of size N_{dd} is done to obtain $\tilde{f}_{\Phi(X;Y)}$, a PDD in the form of a SoDD distribution of size N_{dd}^2, given by:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

[0107] Then, a "compacted" third tuple $R_{FDR}^{N}(\tilde{z})$ may be generated from this PDD $\tilde{f}_{\Phi(X;Y)}$, such that;

$$\mathcal{R}_{FDR}^{N}(\Phi(X,Y))$$

[0108] This is the arithmetically-propagated N-dimensional representation for the random variable;

$$Z=\Phi(X,Y)$$

which is found as the N-dimensional representation of the random variable $\tilde{Z}$ described by the PDD $\tilde{f}_{\Phi(X;Y)}$, that is:

$$\mathcal{R}_{FDR}^{N}(\Phi(X,Y) := \mathcal{R}_{FDR}^{N}(\tilde{Z}).$$

[0109] This equation is where the larger M-tuple representation is reduced back to an N-tuple. This may preferably be done to enable a fixed size for the tuples that one calculates with. Thus, to enable further calculations with the derived tuples (the M-tuples above) one may reduce these M-tuples back to N-tuples. This may be achieved by considering the data represented by $\tilde{f}_{\Phi(X;Y)}$ as being a new "obtained" data set (i.e. an obtained third set of data items, obtained as the result of the arithmetic operation) and therefrom generating a "compacted" third tuple, $R_{FDR}^{N}(\tilde{z})$, containing parameters encoding a probability distribution characterising the distribution of the data items of the third set defined by $f^{\Phi(X;Y)}$.

Arithmetic Operations

[0110] For example, in the case of arithmetic on two RQHR represtations, each defined as a respective ordered N-tuple:

$$\mathcal{R}_{RQHR}^{N}(X) := (x_0, l_X, p_1, \ldots, p_{N_{dd}})$$

the corresponding approximating SoDD distribution for the RQHR in has the form:

$$\tilde{f}_X^{RQHR}(x) := \sum_{n=1}^{N_{dd}} p_n \delta\left(x - \left(x_0 + \left(n - \frac{1}{2}\right) l_X\right)\right)$$

[0111] Of course, for the distribution in the variable Y we simply rewrite the above equation with the substitutions: X→Y and x→y and $x_0 \to y_0$ and $l_X \to l_Y$. Thus, the x_n, y_m within $\Phi(x_n, y_m)$ of the propagation result, $\tilde{f}_{\Phi(X;Y)}(z)$, are given by:

$$x_n = x_0 + \left(n - \frac{1}{2}\right) l_X$$

$$y_m = y_0 + \left(m - \frac{1}{2}\right) l_Y$$

[0112] In the case of the addition of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{+}(x_n, y_m) = x_n + y_m$$

[0113] In the case of the multiplication of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{\times}(x_n, y_m) = x_n \times y_m$$

[0114] For example, in the case of arithmetic on two PQHR represtations, each defined as a respective ordered N-tuple:

$$\mathcal{R}^{N}_{PQHR}(X) := \left(\mu_{X_1}, \ldots, \mu_{X_{N_{dd}}}\right)$$

the corresponding approximating SoDD distribution for the PQHR in has the form:

$$\tilde{f}_X^{PQHR}(x) := \frac{1}{N_{dd}} \sum_{n=1}^{N_{dd}} \delta_{\mu_{X_n}}$$

[0115] Of course, for the distribution in the variable Y we simply rewrite the above equation with the substitutions: $X \rightarrow Y$ and $x \rightarrow y$ and $\mu_{Xn} \Theta \mu_{Ym}$. Thus, the x_n, y_m within $\Phi(x_n, y_m)$ of the propagation result, $\tilde{f}_{\Phi(X;Y)}(z)$, are given by:

$$x_n = \mu_{Xn}$$

$$y_m = \mu_{Ym}$$

[0116] In the case of the addition of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{+}(x_n, y_m) = x_n + y_m$$

[0117] In the case of the multiplication of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{\times}(x_n, y_m) = x_n \times y_m$$

[0118] For example, in the case of arithmetic on two MQHR represtations, each defined as a respective ordered N-tuple:

$$\mathcal{R}^{N}_{MQHR}(X) := \left(x_1, \ldots, x_{N_{dd}}, p_1, \ldots, p_{N_{dd}}\right)$$

the corresponding approximating SoDD distribution for the MQHR has the form:

$$\tilde{f}_X^{MQHR}(x) := \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n)$$

[0119] Of course, for the distribution in the variable Y we simply rewrite the above equation with the substitutions: $X \rightarrow Y$ and $x \rightarrow y$ and $x_n \rightarrow y_m$. Thus, the x_n, y_m within $\Phi(x_n, y_m)$ of the propagation result, $\tilde{f}_{\Phi(X;Y)}(z)$, are given by x_n and y_n of the respective SoDD distributions directly. In the case of the addition of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{+}(x_n, y_m) = x_n + y_m$$

[0120] In the case of the multiplication of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^{\times}(x_n, y_m) = x_n \times y_m$$

[0121] For example, in the case of arithmetic on two TTR represtations, each defined as a respective ordered N-tuple:

$$\mathcal{R}^{N}_{TTR}(X) := \left(\mu_{X_{\varphi_n(1)}}, \ldots, \mu_{X_{\varphi_n(N_{dd})}}, p_{\varphi_n(1)}, \ldots, p_{\varphi_n(N_{dd})}\right)$$

the corresponding approximating SoDD distribution for the TTR has the form:

$$\tilde{f}_X^{TTR}(x) := \sum_{i=1}^{N_{dd}} p_{\varphi_n(i)} \delta\left(x - \mu_{X_{\varphi_n(n)}}\right)$$

[0122] Of course, for the distribution in the variable Y we simply rewrite the above equation with the substitutions: X→Y and x→y and

$$\mu_{X\varphi_n(n)} \to \mu_{Y\varphi_m(m)}.$$

Thus, the x_n, y_m within $\Phi(x_n, y_m)$ of the propagation result, $\tilde{f}_{\Phi(X;Y)}(z)$, are given by:

$$x_n = \mu_{X\varphi_n(n)}$$

$$y_m = \mu_{Y\varphi_m(m)}$$

[0123] In the case of the addition of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^+(x_n, y_m) = x_n + y_m$$

[0124] In the case of the multiplication of distributions, the propagation result is:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n, y_m)).$$

where

$$\Phi(x_n, y_m) = \Phi^\times(x_n, y_m) = x_n \times y_m$$

Propagation of CMRs

[0125] For a given two random variables X and Y the following disclosure describes how one propagates representations of X and Y of given type to find the same type of representation for the random variable Z:=(X; Y). The use of multivariate Taylor expansion of Φ(X; Y) around the expected values of X and Y, i.e., around $(\mu_X; \mu_Y)$ yields:

$$\Phi(x, y) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \frac{\partial^{i+j}}{\partial^i x \partial^j y} \Phi(\mu_X, \mu_Y) \frac{(x - \mu_X)^i (y - \mu_Y)^j}{i! j!}.$$

[0126] For the case of:

$$Z^+ = \Phi^+(X, Y) = X + Y$$

and

$$Z^\times = \Phi^\times\{X, Y\} = X \times Y$$

[0127] The above Taylor expansion reduces to:

$$\Phi^+(x, y) = \mu_X + \mu_Y + (x - \mu_X) + (y - \mu_Y)$$

and

$$\Phi^\times(x, y) = \mu_X \mu_Y + \mu_Y(x - \mu_X) + \mu_X(y - \mu_Y) + (x - \mu_X)(y - \mu_Y)$$

[0128] in the case of mutually independent random variables, i.e., $f_{X;Y}=f_X f_Y$. This result is used to generate the value of $\Phi(x_n,y_m)$ (e.g., $\Phi(X;Y)=\Phi^+(x,y)$ or $\Phi(X;Y)=\Phi^\times(x, y)$ as appropriate) appearing within the function $\tilde{f}_{\Phi(X;Y)}(z)$ defined above.

[0129] For the nth centralized moment of the random variable Z=(X; Y) we have:

$$\mu_Z := \mu_1(Z) := \int_{\mathbb{R}^2} \Phi(x, y) f_{X,Y}(x, y) dxdy,$$

$$\mu_n(Z) := \int_{\mathbb{R}^2} (\Phi(x, y) - \mu_Z)^n f_{X,Y}(x, y) dxdy,$$

$$n \geq 2.$$

[0130] Approximation of this integral in terms of the centralized moments of X and Y uses of multivariate Taylor expansion, defined above, around the expected values of X and Y, i.e., around $(\mu_X; \mu_Y)$. This gives approximations for the first N centralized moments of Z^+ and $Z^\times$ in terms of the first N centralized moments of X and Y as:

$$\begin{aligned}
\mu_{Z^+} &= \mu_1(Z^+) \\
&= \int_{\mathbb{R}^2} \Phi^+(x, y) f_{X,Y}(x, y) dxdy \\
&= \int_{\mathbb{R}^2} [\mu_X + \mu_Y + (x - \mu_X) + (y - \mu_Y)] f_X(x) f_Y(y) dxdy \\
&= \mu_X + \mu_Y, \\
\mu_n(Z^+) &= \int_{\mathbb{R}^2} (\Phi^+(x, y) - \mu_{Z^+})^n f_{X,Y}(x, y) dxdy \\
&= \int_{\mathbb{R}^2} [(x - \mu_X) + (y - \mu_Y)]^n f_X(x) f_Y(y) dxdy \\
&= \sum_{i+j=n} \frac{n!}{i! j!} \left(\int_{\mathbb{R}} (x - \mu_X)^i f_X(x) dx \right) \left(\int_{\mathbb{R}} (y - \mu_Y)^j f_Y(y) dy \right) \\
&= \sum_{\substack{i+j=n \\ i,j \neq 1}} \frac{n!}{i! j!} \mu_i(X) \mu_j(Y),
\end{aligned}$$

$$2 \leq n \leq N.$$

-continued

$$\mu_{Z^{\times}} = \mu_1(Z^{\times})$$

$$= \int_{\mathbb{R}^2} \Phi^{\times}(x, y) f_{X,Y}(x, y) dxdy,$$

$$= \int_{\mathbb{R}^2} [\mu_X \mu_Y + \mu_Y(x - \mu_X) + \mu_X(y - \mu_Y)] + (x - \mu_X)(y - \mu_Y)] f_X(x) f_Y(y) dxdy$$

$$= \mu_X \mu_Y,$$

$$\mu_n(Z^{\times}) = \int_{\mathbb{R}^2} (\Phi^{\times}(x, y) - \mu_{Z^{\times}})^n f_{X,Y}(x, y) dxdy$$

$$= \int_{\mathbb{R}^2} [\mu_Y(x - \mu_X) + \mu_X(y - \mu_Y) + (x - \mu_X)(y - \mu_Y)]^n f_X(x) f_Y(y) dxdy$$

$$= \sum_{i+j+k=n} \frac{n!}{i!j!k!} \mu_Y^i \mu_X^j \left(\int_{\mathbb{R}} (x - \mu_X)^{i+k} f_X(x) dx \right) \left(\int_{\mathbb{R}} (y - \mu_Y)^{j+k} f_Y(y) dy \right)$$

$$= \sum_{\substack{i+j+k=n \\ i+k, j+k \neq 1}} \frac{n!}{i!j!k!} \mu_Y^i \mu_X^j \mu_{i+k}(X) \mu_{j+k}(Y),$$

$$2 \leq n \leq N.$$

[0131] For example, let X and Y be two mutually independent random variables with N-dimensional CMRs:

$$\mathcal{R}^N_{CMR}(X) = (\mu_1(X), \ldots, \mu_N(X)) \text{ and}$$

$$\mathcal{R}^N_{CMR}(Y) = (\mu_1(Y), \ldots, \mu_N(Y))$$

respectively. Then, one can calculate the N-dimensional CMRs:

$$\mathcal{R}^N_{CMR}(Z^{+}) = (\mu_1(Z^{+}), \ldots, \mu_N(Z^{+})) \text{ and}$$

$$\mathcal{R}^N_{CMR}(Z^{\times}) = (\mu_1(Z^{\times}), \ldots, \mu_N(Z^{\times}))$$

of the random variables $Z^{+}=X+Y$ and $Z^{\times}=X\times Y$ exactly via the relations given above, respectively.

[0132] The present invention may encode, represent and propagate distributional information/data (e.g., probability distributions, frequency distributions etc.) representing uncertainty in measurement data made by a measurement apparatus (e.g., sensor) and/or may propagate distributional information/data defining distributional weights of an artificial neural network.

[0133] In general, an arithmetic operation may be performed by using the parameters (e.g., Dirac delta position, height/probability) of the first tuple (i.e., representing a first distribution), and using the parameters (e.g., e.g., Dirac delta position, height/probability) of the second tuple (i.e., representing a second distribution) to generate a third tuple comprising parameters (e.g., e.g., Dirac delta position, height/probability) having values defining the distribution resulting from the arithmetic operation applied to the first distribution and the second distribution. This third tuple may then preferably be used to generate a "compacted" third tuple, as described herein, which has the same size (e.g., an N-tuple) as that of the first and second tuples. The first and second tuples preferably have the same size (e.g., both being an N-tuple).

[0134] The positions of Dirac delta functions representing data of the third distribution may be defined by the position values calculated (e.g., as disclosed herein) using the positions of Dirac delta functions representing data of the first and second data distributions. These position values of the first and second data distributions may be contained within the first and second tuples. The heights/probabilities of Dirac delta functions representing data of the third distribution may be generated by calculating the product of the heights/probabilities of the Dirac delta functions representing data of the first and second data distributions. These height/probability values of the first and second data distributions may be contained within the first and second tuples.

[0135] The third tuple may contain probability values (e.g., Dirac delta height/probability) each generated according to a respective probability value in the first tuple and a respective probability value in the second tuple (e.g., a product of a respective probability value in the first tuple and a respective probability value in the second tuple when the arithmetic operation is a multiplication operation or an addition operation etc.). The probability values may represent an amplitude, height or weighting of a Dirac delta function within a distribution represented by the third tuple.

[0136] The third tuple may contain position values each generated according to a respective position value in the first tuple and a respective position value in the second tuple (e.g., a product (or an addition) of a respective position value in the first tuple and a respective position value in the second tuple when the arithmetic operation is a multiplication operation (or an addition operation) etc.). The position values may represent a position of a Dirac delta function within a distribution represented by the third tuple.

[0137] The invention is not limited to Dirac delta function-based representations (such as MQHR), and the first, second and third tuples may generally contain parameters, according to other representations, e.g., encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the first, second and third distributions. For example, the first, second and third tuples may contain parameters encoding the value of one or more statistical moments of a probability distribution characterising the distribution of data items.

[0138] The arithmetic process allows each parameter of the third tuple to be generated using the parameters of the first tuple and the second tuple. Once the parameters of the third tuple are calculated, then this fully encodes the distributional information of the third distribution, permitting that third distribution to be reproduced as and when required, and permitting the third tuple to be stored in a memory medium and/or transmitted as a signal, in a very efficient form.

[0139] In other aspects, either separately or in conjunction with any other aspect herein, the invention may concern the following aspects. In other words, any one or more of the aspects described below may be considered as applying separately from, or in combination with, any of the aspects described above. For example, the apparatus described above may comprise the microarchitecture described below, and similarly so for the methods described above and the methods described below.

[0140] In a third aspect, the invention may provide a method for computation on distributions of data, the method comprising providing a microarchitecture comprising:

[0141] a first register containing data items;

[0142] a second register containing distribution data representing distributions that are uncertainty representations associated with respective said data items;

[0143] a first arithmetic logic unit for executing arithmetic on data items selected from the first register;

[0144] a second arithmetic logic unit for executing arithmetic on distribution data selected from the second register;

[0145] the method comprising the following steps implemented by the microarchitecture:

[0146] executing, by the first arithmetic logic unit, an arithmetic operation on data items selected from the first register, and outputting the result;

[0147] executing, by the second arithmetic logic unit, an arithmetic operation on distribution data representing distributions selected from the second register that are associated with the data items selected from the first register, and outputting the result;

[0148] wherein the arithmetic operation executed on the distribution data selected from the second register is the same as the arithmetic operation executed on the data items selected from the first register thereby to generate further distribution data representing uncertainty associated with the result of the arithmetic operation executed on the data items selected from the first register.

[0149] In this way, a first register (e.g. item 'f0' of FIG. **6**A) may contain data that may be a single item. The second register (e.g. item 'df0' of FIG. **6**A) may contain distribution data of uncertainty representations of the single item contained in the first register. References herein to a first register containing data items may be considered to include a reference to a first set of registers containing data items. Similarly, references herein to a second register containing distribution data representing distributions may be considered to include a reference to a second set of registers containing distribution data representing distributions. A set of registers may be considered to be a register file.

[0150] Preferably, the first register may comprise a first set of registers, and/or may comprise a register file. Preferably, the second register may comprise a first set of registers, and/or may comprise a register file.

[0151] In the method, the microarchitecture may comprise, for example, a floating-point register file configured to contain floating-point data. The floating-point register file may contain a first register file configured for containing particle data items, and a second register file configured for containing distribution data. The microarchitecture may comprise, for example, an integer register file configured to contain integer data. The integer register file may contain a first register file configured for containing particle data items, and a second register file configured for containing distribution data. The distributional data may represent distributions (e.g. SoDD-based representations) that are uncertainty representations associated with respective "particle" data items in the first register file. In the method, the floating-point register file may associate a given "particle" data item in the first register file with its associated distributional data within the second register file by assigning one common register file entry identifier to both a given "particle" data value within the first register file and the distributional data entry in the second register file that is to be associated with the "particle" data in question. In this way, the floating-point and/or integer register files in the microarchitecture may associate all floating-point and/or integer registers with distributional information.

[0152] The microarchitecture, and the invention in general, may calculate more complex arithmetic operations e.g., fused multiplication and addition or square root, by expressing them in terms of the aforementioned basic arithmetic operations, as would be readily apparent the skilled person.

[0153] Desirably, the execution of the arithmetic operation by the second arithmetic logic unit is triggered by a command that triggers the execution of the arithmetic operation by the first arithmetic logic unit. The arithmetic operation preferably comprises one or more of: addition; subtraction; multiplication; division. The invention in general may calculate more complex arithmetic operations e.g., fused multiplication and addition or square root, by expressing them in terms of the aforementioned basic arithmetic operations, as would be readily apparent to the skilled person.

[0154] The method may include providing said microarchitecture comprising a memory unit configured to store said data items at addressed memory locations therein, the method comprising the following steps implemented by the microarchitecture: obtaining the originating memory location addresses of data items that contribute to the arithmetic operation executed by the first arithmetic logic unit as the first arithmetic logic unit executes the arithmetic operation; and, storing the obtained originating memory location addresses at a storage location within the memory unit and associating the storage location with said further distribution data. The obtained originating memory locations may be stored in a combined form, such that multiple originating memory location addresses may be stored together, e.g., in a table, with multiple originating memory location addresses stored in the same table entry/location.

[0155] The results of arithmetic operations may be output to a random-access memory. These output results may comprise both "particle" data values and distributional data. Each "particle" data value may be stored in a memory unit, providing a physical address space, and may be associated with a distribution representation stored in a distributional memory unit. A memory access unit and a register writeback unit may be provided to define an interface between the register files and the arithmetic logic units of the microarchitecture. The Instruction Fetch unit may be provided in communication with the "particle" memory unit for accessing the memory unit for fetching instructions therefrom. A Load/Store unit may be provided to be in direct communication with the distributional memory unit, and an Instruction Fetch unit may be provided but not so connected. This means that the execution of arithmetic operations on distributional data may take place automatically without requiring, or interfering with, the operation of the Instruction Fetch unit. In this way, the calculation of distributional data, resulting from arithmetic operations on "particle" data and their associated distributional data, may take place at the microarchitectural level. Accordingly, the microarchitecture may be configured to allow only load/store instructions to access the random-access memory. Consequently, an extended memory may be provided in this way to which the microarchitecture can load and store both the "particle" and distributional information of the microarchitecture registers.

[0156] In the method, the microarchitecture may be configured to track which memory addresses of a memory unit have contributed to the calculation of the value of any given floating-point or integer register at any point in time. When a "particle" value resulting from an arithmetic operation, is output from a register of the microarchitecture, the output data of a particle value resulting from an arithmetic operation may be stored in memory. The information about the original addresses, or originating addresses, of the particle data items that contributed to the output result (referred to herein as "origin addresses" or "ancestor addresses": these two terms refer to the same thing) may also be stored within a memory unit. The processor may be configured to subsequently recall the origin addresses when the contents of the register (e.g., the stored "particle" value) are loaded from memory for further use. This is discussed in more detail below and we refer to this correlation tracking as the "origin addresses tracking" mechanism. In the present invention, in preferred embodiments, the value of each floating-point/integer register originates from one or more address of the memory unit. By maintaining and propagating these addresses the invention is able to dynamically identify correlations between any two floating-point/integer registers. This information may be maintained, for example, using a dynamically linked list of "origin addresses".

[0157] In the method, the first register preferably contains a first data item and a second data item. For example, the first register may comprise a first set of registers containing data items, which include the first data item and the second data item. Preferably, the first data item comprises a value of a first random variable, and the second data item comprises a value of a second random variable. Similarly, the second register may comprise a second set of registers containing distribution data representing distributions. For example, the first and second registers may each be a register file. Desirably, the second register contains first distribution data comprising a first tuple containing parameters encoding a probability distribution characterising the uncertainty representation associated with the first data item. Preferably, the second register contains second distribution data comprising a second tuple containing parameters encoding a probability distribution characterising the uncertainty representation associated with the second data item in which the parameters used to encode the second distribution data are the same as the parameters used to encode the first distribution data.

[0158] Desirably, said executing, by the second arithmetic logic unit, an arithmetic operation on distribution data comprises selecting the first tuple and the second tuple and therewith generating a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding said further distribution data. The method may comprise outputting the third tuple. The outputting of the tuple may comprise outputting it to a memory for storage, or outputting to a transmitter for transmission.

[0159] Preferably, the third tuple contains parameters encoding said further distribution data that are the same as the parameters used to encode the probability distribution characterising the uncertainty representation associated with the first data item. Of course, if the second tuple contains parameters encoding a probability distribution that are the same as the parameters used in the first tuple to encode the first distribution data, as discussed above, then these parameters will also be the same parameters (i.e. "same" in nature, but generally not "same" in value) as used in the third tuple.

[0160] Desirably, the distribution data comprises probability distributions of respective data Items.

[0161] Preferably, the first tuple contains parameters encoding the position of data items within the probability distribution characterising the uncertainty representation associated with the first data item. Desirably, the second tuple contains parameters encoding the position of data items within the probability distribution characterising the uncertainty representation associated with the second data item. Preferably, the third tuple contains parameters encoding the position of data items within a probability distribution characterising the further distribution data.

[0162] Desirably, the first tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the uncertainty representation associated with the first data item. Preferably, the second tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the uncertainty representation associated with the second data item. Desirably, the third tuple contains parameters encoding the position and/or width of data intervals within a probability distribution characterising the further distribution data.

[0163] Preferably the first tuple contains parameters encoding the probability of data items within the probability distribution characterising the uncertainty representation associated with the first data item. Desirably, the second tuple contains parameters encoding the probability of data items within the probability distribution characterising the uncertainty representation associated with the second data item. Preferably, the third tuple contains parameters encoding the probability of data items within a probability distribution characterising the further distribution data.

[0164] Preferably, the first tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the uncertainty representation associated with the first data item. Desirably, the second tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the uncertainty representation associated with the second data item. Preferably, the third tuple contains parameters encoding the value of one or more statistical moments of a probability distribution characterising the further distribution data.

[0165] Preferably, the probability distribution characterising the uncertainty representation associated with the first data item comprises a distribution of Dirac delta functions. Desirably, the probability distribution characterising the uncertainty representation associated with the second data item comprises a distribution of Dirac delta functions. Preferably, the probability distribution characterising the further distribution data comprises a distribution of Dirac delta functions.

[0166] Preferably, the first tuple is an N-tuple in which $N>1$ is an integer. Preferably, the second tuple is an N-tuple in which $N>1$ is an integer. Desirably, the third tuple is an M-tuple for which $N^2/2<M<2N^2$ in which $N>1$ is an integer. The outputting of the results from the first and/or second arithmetic logic unit may comprise one or more of: storing the output in a memory; transmitting a signal conveying the output.

[0167] In another aspect, the invention may provide a computer program product comprising a computer program which, when executed on a computer, implements the method described above in the third aspect of the invention. In yet another aspect, the invention may provide a computer programmed with a computer program which, when executed on the computer, implements the method described above in the third aspect of the invention.

[0168] In a fourth aspect, the invention may provide a microarchitecture for computation on distributions of data comprising:

[0169] a first register configured for containing data items;

[0170] a second register configured for containing distribution data representing distributions that are uncertainty representations associated with respective said data items;

[0171] a first arithmetic logic unit configured for executing arithmetic on data items selected from the first register;

[0172] a second arithmetic logic unit configured for executing arithmetic on distribution data selected from the second register;

the microarchitecture configured to implement the following steps:

[0173] executing, by the first arithmetic logic unit, an arithmetic operation on data items selected from the first register, and outputting the result;

[0174] executing, by the second arithmetic logic unit, an arithmetic operation on distribution data representing distributions selected from the second register that are associated with the data items selected from the first register, and outputting the result;

[0175] wherein the arithmetic operation executed on the distribution data selected from the second register is the same as the arithmetic operation executed on the data items selected from the first register thereby to generate further distribution data representing uncertainty associated with the result of the arithmetic operation executed on the data items selected from the first register.

[0176] Preferably, the first register may comprise a first set of registers, and/or may comprise a register file. Preferably, the second register may comprise a first set of registers, and/or may comprise a register file.

[0177] References herein to a first register containing data items may be considered to include a reference to a first set of registers containing data items. Similarly, references herein to a second register containing distribution data representing distributions may be considered to include a reference to a second set of registers containing distribution data representing distributions. A set of registers may be considered to be a register file.

[0178] The microarchitecture may comprise, for example, a floating-point register file configured to contain floating-point data. The floating-point register file may contain a first register file configured for containing particle data items, and a second register file configured for containing distribution data. The microarchitecture may comprise, for example, an integer register file configured to contain integer data. The integer register file may contain a first register file configured for containing particle data items, and a second register file configured for containing distribution data. The distributional data may represent distributions (e.g. SoDD-based representations) that are uncertainty representations associated with respective "particle" data items in the first register file.

[0179] The floating-point register file may associate a given "particle" data item in the first register file with its associated distributional data within the second register file by assigning one common register file entry identifier to both a given "particle" data value within the first register file and the distributional data entry in the second register file that is to be associated with the "particle" data in question. In this way, the floating-point and/or integer register files in the microarchitecture may associate all floating-point and/or integer registers with distributional information.

[0180] The microarchitecture maybe configured to execute an arithmetic operation comprising one or more of: addition; subtraction; multiplication; division, or more complex arithmetic operations e.g., fused multiplication and addition or square root, or any bivariate operation, e.g., exponentiation, by expressing them in terms of the aforementioned basic arithmetic operations, as would be readily apparent to the skilled person.

[0181] The microarchitecture may be configured to output the results of arithmetic operations to a random-access memory. These output results may comprise both "particle" data values and distributional data. The microarchitecture may be configured to store each "particle" data value in a memory unit, providing a physical address space, and may be associated with a distribution representation stored in a distributional memory unit. The microarchitecture may comprise a memory access unit and a register writeback unit to define an interface between the register files and the arithmetic logic units of the microarchitecture. The microarchitecture may comprise a Instruction Fetch unit configured in communication with the "particle" memory unit for accessing the memory unit for fetching instructions therefrom. The microarchitecture may comprise a Load/Store unit configured to be in direct communication with the distributional memory unit, and the microarchitecture may comprise an Instruction Fetch unit not so connected. This means that the execution of arithmetic operations on distributional data may take place automatically without requiring, or interfering with, the operation of the Instruction Fetch unit. Accordingly, the microarchitecture may be configured to allow only load/store instructions to access the random-access memory. Consequently, the microarchitecture may be configured to load and store both the "particle" and distributional information of the microarchitecture registers.

[0182] The microarchitecture may be configured to track which memory addresses of a memory unit have contributed to the calculation of the value of any given floating-point or integer register at any point in time. When a "particle" value resulting from an arithmetic operation, is output from a register of the microarchitecture, the output data of a particle value resulting from an arithmetic operation may be stored in memory. The information about the original addresses, or originating addresses, of the particle data items that contributed to the output result (referred to herein as "origin addresses" or "ancestor addresses": these two terms refer to the same thing) may also be stored within a memory unit. The processor may be configured to subsequently recall the origin addresses when the contents of the register (e.g., the stored "particle" value) are loaded from memory for further use. This is discussed in more detail below and we refer to

this correlation tracking as the "origin addresses tracking" mechanism. In the present invention, in preferred embodiments, the value of each floating-point/integer register originates from one or more address of the memory unit. By maintaining and propagating these addresses the invention is able to dynamically identify correlations between any two floating-point/integer registers. This information may be maintained, for example, using a dynamically linked list of "origin addresses".

[0183] The microarchitecture may be configured to execute the arithmetic operation by the second arithmetic logic unit when triggered by a command that triggers the execution of the arithmetic operation by the first arithmetic logic unit. The outputting by the microarchitecture may comprise one or more of: storing the output in a memory; transmitting a signal conveying the output.

[0184] The microarchitecture may comprise a memory unit configured to store said data items at addressed memory locations therein. The microarchitecture is preferably configured to obtain the originating memory location addresses of data items that contribute to the arithmetic operation executed by the first arithmetic logic unit as the first arithmetic logic unit executes the arithmetic operation. The microarchitecture is preferably configured to store the obtained originating memory location addresses at a storage location within the memory unit and associating the storage location with said further distribution data. The obtained originating memory locations may be stored in a combined form, such that multiple originating memory location addresses may be stored together, e.g. in a table, with multiple originating memory location addresses stored in the same table entry/location.

[0185] The first register is preferably configured to contain a first data item and a second data item. For example, the first register may comprise a first set of registers containing data items, which include the first data item and the second data item. Preferably, the first data item comprises a value of a first random variable, and the second data item comprises a value of a second random variable. Similarly, the second register may comprise a second set of registers containing distribution data representing distributions. For example, the first and second registers may each be a register file. Preferably, the second register is configured to contain first distribution data comprising a first tuple containing parameters encoding a probability distribution characterising the uncertainty representation associated with the first data item. Desirably, the second register is configured to contain second distribution data comprising a second tuple containing parameters encoding a probability distribution characterising the uncertainty representation associated with the second data item in which the parameters used to encode the second distribution data are the same as the parameters used to encode the first distribution data.

[0186] The microarchitecture may be configured to execute, by the second arithmetic logic unit, an arithmetic operation on distribution data comprising selecting the first tuple and the second tuple and therewith generating a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding said further distribution data. The microarchitecture may be configured to output the third tuple.

[0187] Desirably, the third tuple contains parameters encoding said further distribution data that are the same as the parameters used to encode the probability distribution characterising the uncertainty representation associated with the first data item.

[0188] Preferably, the distribution data comprises probability distributions of respective data items.

[0189] The first tuple may contain parameters encoding the position of data items within the probability distribution characterising the uncertainty representation associated with the first data item. The second tuple may contain parameters encoding the position of data items within the probability distribution characterising the uncertainty representation associated with the second data item. The third tuple may contain parameters encoding the position of data items within a probability distribution characterising the further distribution data.

[0190] The first tuple may contain parameters encoding the position and/or width of data intervals within the probability distribution characterising the uncertainty representation associated with the first data item. The second tuple may contain parameters encoding the position and/or width of data intervals within the probability distribution characterising the uncertainty representation associated with the second data item. The third tuple may contain parameters encoding the position and/or width of data intervals within a probability distribution characterising the further distribution data.

[0191] The first tuple may contain parameters encoding the probability of data items within the probability distribution characterising the uncertainty representation associated with the first data item. The second tuple may contain parameters encoding the probability of data items within the probability distribution characterising the uncertainty representation associated with the second data item. The third tuple may contain parameters encoding the probability of data items within a probability distribution characterising the further distribution data.

[0192] The first tuple may contain parameters encoding the value of one or more statistical moments of the probability distribution characterising the uncertainty representation associated with the first data item. The second tuple may contain parameters encoding the value of one or more statistical moments of the probability distribution characterising the uncertainty representation associated with the second data item. The third tuple may contain parameters encoding the value of one or more statistical moments of a probability distribution characterising the further distribution data.

[0193] The probability distribution characterising the uncertainty representation associated with the first data item may comprise a distribution of Dirac delta functions. The probability distribution characterising the uncertainty representation associated with the second data item comprises a distribution of Dirac delta functions. The probability distribution characterising the further distribution data comprises a distribution of Dirac delta functions.

[0194] The first tuple may be an N-tuple, in which $N>1$ is an integer. The second tuple may be an N-tuple, in which $N>1$ is an integer. The third tuple may be an M-tuple for which $N^2/2<M<2N^2$ in which $N>1$ is an integer.

[0195] In other aspects, either separately or in conjunction with any other aspect herein, the invention may concern the following aspects. In other words, any one or more of the aspects described below may be considered as applying separately from, or in combination with, any of the aspects

described above. For example, the apparatus described above may comprise the apparatus described below, and similarly so for the methods described above and the methods described below.

[0196] In another aspect, either separately or in conjunction with any other aspect herein, the invention may concern a method that can be implemented in software of within the hardware of a microprocessor, FPGA, or other digital computation device, for representing probability distributions of items including both numeric values (e.g., integers and floating-point approximate real numbers) as well as categorical values (e.g., sets of items that have no numeric interpretation). The invention may represent both distributions in the traditional statistical sense as well as set-theoretic collections where the elements of the set have different probabilities of membership. In the special case where the values in the sets are integers or floating-point numbers, the invention can be used as the data representation for a computing system that performs computation natively on probability distributions of numbers in the same way that a traditional microprocessor performs operations on integers and floating-point values, and methods and apparatus for executing arithmetic on distributions disclosed herein may be used.

[0197] In other aspects, either separately or in conjunction with any other aspect herein, the invention may concern the following aspects. In other words, any one or more of the aspects described below may be considered as applying separately from, or in combination with, any of the aspects described above. For example, the apparatus described above may comprise the apparatus described below, and similarly so for the methods described above and the methods described below.

[0198] In a fifth aspect, the invention may provide a computer-implemented method for the encoding of distributions of data, the method comprising:

[0199] obtaining a set of data items;

[0200] determining a probability (or frequency) distribution for the obtained data items;

[0201] selecting a sub-set of the data items within the obtained data set having a probability e.g. an aggregate probability of occurrence within the obtained data set which exceeds the value of a pre-set threshold probability or which comprise a pre-set number of data items having the highest probability of occurrence within the obtained data set;

[0202] generating a tuple for each of the data items within the sub-set whereby a respective tuple comprises a first value and a second value wherein the first value is a value of the respective data item and the second value is a value of the probability of occurrence of the respective data item within the obtained data set;

[0203] normalising the probability of occurrence of each respective selected data item such that the sum of said probabilities of occurrence of all selected data items is equal to 1.0;

[0204] providing a memory and for each tuple storing therein the first value thereof at a respective memory location;

[0205] providing a table (in software or hardware) and for each tuple storing therein the second value thereof in association with a pointer configured to identify the respective memory location of the first value thereof.

[0206] It is to be understood that the step of generating a tuple for each of the data items of the sub-set may be implemented either after or before the step of selecting a sub-set of data items. If is implemented after then the subsequent step of generating the tuples comprises generating tuples for only those data items within the sub-set. If is implemented before then the preceding/prior step of generating the tuples comprises generating tuples for all data items of the obtained data set and then the subsequent step of selecting the sub-set of data items proceeds by simply selecting those tuples possessing a value of probability which exceeds the value of a pre-set threshold probability or which define a pre-set number of tuples having the highest probability values amongst all of the tuples generated for the data set. In this way, the sub-set of data items may be selected via the tuples representing them, by proxy.

[0207] The set of data items may comprise a set of numeric values or may comprise a set of categorical values, or may represent value ranges (e.g. value 'bins' in a distribution or histogram). The set of data items may be obtained from a measurement apparatus (e.g. a sensor), or may be obtained from the output of a machine learning model (e.g., weights of an artificial neural network etc.), or may be obtained from the output of a quantum computer for use in a classical computer when the measurement of the state of qubits of the quantum computer collapses from their superposed states to measured values of "1" or "0" with associated probabilities (e.g. a vector of a Bernoulli random variables). The set of data items may be obtained from a database or memory store of data items (numeric or categorical).

[0208] The step of determining a probability distribution may comprise determining a statistical data set (or a population) comprising a listing or function showing all the possible values (or intervals, ranges/bins etc.) of the data and how often they occur. The distribution may be a function (empirical or analytical) that shows values for data items within the data set and how often they occur. A probability distribution may be the function (empirical or analytical) that gives the probabilities of occurrence of different possible/observed values of data items of the data set.

[0209] The method may include determining a probability of occurrence of the data items within the obtained data set which do not exceed the value of the pre-set threshold probability (or threshold frequency), or which are not amongst the pre-set number. In other words, the method may include determining the further probability of occurrence of data items within the obtained data set that do not belong to the sub-set. The step of normalising the probability of occurrence of each respective selected data item within the sub-set may be such that the sum of the further probability and said probabilities of occurrence of all selected data items is equal to 1.0. In this way, the further probability may take account of the probability of a data item of the obtained data set being outside of the sub-set and thereby provide an 'overflow' probability.

[0210] The method may include generating a collective tuple for data items of the obtained data set not within the sub-set comprising a first value and a second value wherein the first is a value collectively representative of the data items of the obtained data set not within the sub-set and the second value is a value of the normalised further probability of occurrence of the data items within the obtained data set, but not within the sub-set.

[0211] The method may include storing the first value of the collective tuple in said memory at a respective memory location, and storing in said table the second value of the collective tuple in association with a pointer configured to identify the respective memory location of the first value of the collective tuple.

[0212] The method may include generating a representation of the distribution probability distribution for the sub-set of data items using the first value and the second value of a plurality of the aforesaid tuples of the sub-set (e.g., using all of them). The method may include generating a representation of the distribution probability distribution for the obtained data items using the first value and the second value of a plurality of the aforesaid tuples of the sub-set and using the first value and the second value of the collective tuple.

[0213] A representation of the distribution probability distribution may be according to any representation disclosed herein (e.g., a SoDD-based representation). The method may include performing an arithmetic operation on two distributions wherein one or both of the distributions are generated as described above.

[0214] The aforesaid table may be a structured memory in a circuit structure rather than an array in a computer memory. The method may include providing a logic unit configured for receiving said tuples (e.g. of the sub-set, and optionally also the collective tuple) and for storing the second values thereof, in association with respective pointers, at locations within a sub-table of the table if the first values of the tuples comply with criteria defined by the logic unit. In this way, the method may structure the table to comprise sub-tables each of which contains data from tuples satisfying a particular set of criteria. The criteria defining any one of the sub-tables may, of course, be different to the criteria defining any of the other sub-tables.

[0215] The method may comprise providing a copula structure for combining the individual distributions represented using the tuples of separate tables (each formed as described above), or of separate sub-tables, to achieve joint distributions. The copula structure may be configured to generate a multivariate cumulative distribution function for which the marginal probability distribution of each variable is uniform on the interval [0, 1]. Copulas are used to describe the dependence between random variables. Copulas allow one to model and estimate the distribution of random variables by estimating marginals and copulae separately. There are many parametric copula families available to the skilled person for this purpose.

[0216] The logic unit may be configured to implement a probabilistic (e.g. non-Boolean) predicate for the table which is configured to return a probability (p) (i.e. rather than a Boolean 'true/'false' value), for each element of the table corresponding to a said first value (i.e. a value of a respective data item). The criteria defining different sub-tables may be implemented in this way. Since the predicate may be considered in terms of a predicate tree, this permits the predicate to be flattened into a string using pre-order traversal or post-order traversal of all of the nodes of the tree and this flattened tree can be used as the distribution representation, if desired. This may significantly reduce the size of the representation to being one that scales linearly with the size of the representation rather than exponentially. Techniques for pre-order traversal or post-order traversal of all of the nodes of the tree may be according to techniques readily available to the skilled person.

[0217] A tuple may take the form of a data structure consisting of multiple parts; an ordered set of data, e.g. constituting a record. References herein to a "tuple" may be considered to include a reference to a finite ordered list of elements. References herein to an "n-tuple" may be considered to include a reference to a sequence of n elements, where n is a non-negative integer.

[0218] In a sixth aspect, the invention may provide an apparatus for the encoding of distributions of data, the apparatus being configured to implement the following steps, comprising:

[0219] obtaining a set of data items;

[0220] determining a probability distribution for the obtained data items;

[0221] selecting a sub-set the data items within the obtained data set having a probability (e.g. aggregate probability) of occurrence within the obtained data set which exceeds the value of a pre-set threshold probability or which comprise a pre-set number of data items having the highest probability of occurrence within the obtained data set;

[0222] generating a tuple for each of the data items within the sub-set whereby a respective tuple comprises a first value and a second value wherein the first value is a value of the respective data item and the second value is a value of the probability of occurrence of the respective data item within the obtained data set;

[0223] normalising the probability of occurrence of each respective selected data item such that the sum of said probabilities of occurrence of all selected data items is equal to 1.0;

[0224] providing a memory and for each tuple storing therein the first value thereof at a respective memory location;

[0225] providing a table (in software or hardware) and for each tuple storing therein the second value thereof in association with a pointer configured to identify the respective memory location of the first value thereof.

[0226] The apparatus may be configured to implement the step of determining a probability distribution by determining a statistical data set (or a population) comprising a listing or function showing all the possible values (or intervals, ranges/bins etc.) of the data and how often they occur. Preferably, the apparatus may be implemented as a microprocessor, or a dedicated digital logic circuit, or an analogue circuit configured to perform the processing steps.

[0227] The apparatus may be configured to determine a probability of occurrence of the data items within the obtained data set which do not exceed the value of the pre-set threshold probability or are not amongst the pre-set number. The apparatus may be configured to implement the step of normalising the probability of occurrence of each respective selected data item within the sub-set may be such that the sum of the further probability and said probabilities of occurrence of all selected data items is equal to 1.0.

[0228] The apparatus may be configured to generate a collective tuple for data items of the obtained data set not within the sub-set comprising a first value and a second value wherein the first is a value collectively representative of the data items of the obtained data set not within the sub-set and the second value is a value of the normalised

further probability of occurrence of the data items within the obtained data set, but not within the sub-set.

[0229] The apparatus may be configured to store the first value of the collective tuple in said memory at a respective memory location, and store in said table the second value of the collective tuple in association with a pointer configured to identify the respective memory location of the first value of the collective tuple.

[0230] The apparatus may be configured to generate a representation of the distribution probability distribution for the sub-set of data items using the first value and the second value of a plurality of the aforesaid tuples of the sub-set (e.g., using all of them). The apparatus may be configured to generate a representation of the distribution probability distribution for the obtained data items using the first value and the second value of a plurality of the aforesaid tuples of the sub-set and using the first value and the second value of the collective tuple.

[0231] The apparatus may be configured with a structured memory in a circuit structure rather than an array in a computer memory. The apparatus may provide a logic unit configured for receiving said tuples (e.g. of the sub-set, and optionally also the collective tuple) and for storing the second values thereof, in association with respective pointers, at locations within a sub-table of the table if the first values of the tuples comply with criteria defined by the logic unit.

[0232] The apparatus may provide a copula structure for combining the individual distributions represented using the tuples of separate tables (each formed as described above), or of separate sub-tables, to achieve joint distributions. The logic unit may be configured to implement a probabilistic (e.g., non-Boolean) predicate for the table which is configured to return a probability (p) (i.e. rather than a Boolean 'true/false' value), for each element of the take corresponding to a said first value (i.e. a value of a respective data item). The criteria defining different sub-tables may be implemented in this way.

[0233] In another aspect, the invention may provide a computer program product comprising a computer program which, when executed on a computer, implements the method according to the invention described above. In another aspect, the invention may provide a computer programmed with a computer program which, when executed on the computer, implements the method according described above.

[0234] References herein to "threshold" may be considered to include a reference to a value, magnitude or quantity that must be equalled or exceeded for a certain reaction, phenomenon, result, or condition to occur or be manifested.

[0235] References herein to "distribution" in the context of a statistical data set (or a population) may be considered to include a reference to a listing or function showing all the possible values (or intervals) of the data and how often they occur. A distribution in statistics may be thought of as a function (empirical or analytical) that shows the possible values for a variable and how often they occur. In probability theory and statistics, a probability distribution may be thought of as the function (empirical or analytical) that gives the probabilities of occurrence of different possible outcomes for measurement of a variable.

[0236] The invention includes the combination of the aspects and preferred features described except where such a combination is clearly impermissible or expressly avoided.

[0237] In other aspects, either separately or in conjunction with any other aspect herein, the invention may concern the following aspects. In other words, any one or more of the aspects described below may be considered as applying separately from, or in combination with, any of the aspects described above. For example, the apparatus described above may comprise the apparatus described below, and similarly so for the methods described above and the methods described below.

[0238] In other aspects, either separately or in conjunction with any other aspect herein, the invention may concern the following aspects. In other words, any one or more of the aspects described below may be considered as applying separately from, or in combination with, any of the aspects described above. For example, the apparatus described above may comprise the apparatus described below, and similarly so for the methods described above and the methods described below.

[0239] In another aspect, either separately or in conjunction with any other aspect herein, the invention may concern rearranging instruction sequences to reduce numeric error in propagating uncertainty across the computation state (e.g., registers) on which the instruction sequences operate.

[0240] In a seventh aspect, the invention may provide a computer-implemented method for computing a numerical value for uncertainty in the result of a multi-step numerical calculation comprising a sequence of separate calculation instructions defined within a common "basic block", the method comprising:

[0241] (a) identifying a "live-out" variable of the "basic block";

[0242] (b) identifying calculation instructions on who's output the value of the "live-out" variable depends;

[0243] (c) providing a mathematical expression combining the calculation instructions identified at step (b);

[0244] (d) using the mathematical expression of provided at step (c) to compute a numerical value for uncertainty in the "live-out" variable;

wherein the uncertainty value computed at step (d) is the uncertainty in the result of the multi-step numerical calculation. It has been found that this process not only makes the final result of the calculation more accurate, but also makes the calculation process more efficient.

[0245] Preferably, in the method, for the live-out variables of a basic block, rather than computing the updated uncertainty for each instruction on whose results they depend, the sequence of instructions whose result determines the value of the live-out variable can be combined into a single expression for the purposes of computing the updated uncertainty of the live-out variable.

[0246] A reference to a variable as "live-out" may be considered to include a reference to a variable being "live-out" at a node (e.g. of an instruction sequence) if it is live on any of the out-edges from that node. A reference to a variable as "live-out" may be considered to include a reference to a variable being a live register entry (e.g. a register whose value will be read again before it is overwritten).

[0247] A reference to a "basic block" may be considered to include a reference to a sequence of instructions with no intervening control-flow instruction. The "live-in" variables for a basic block may be considered to be the program variables or machine registers whose values will be read before they are overwritten within the basic block. The "live-out" variables for a basic block may be considered to

be the program variables or machine registers whose values will be used after the basic block exits. For the live-out variables of a basic block, rather than computing the updated uncertainty for each instruction on whose results they depend, the sequence of instructions whose result determines the value of the live-out variable can be combined into a single expression for the purposes of computing the updated uncertainty of the live-out variable.

[0248] In another aspect, the invention may provide a computer program product comprising a computer program which, when executed on a computer, implements the method according to the invention described above, in its first aspect.

[0249] In another aspect, the invention may provide a computer programmed with a computer program which, when executed on the computer, implements the method according described above, in the first aspect of the invention.

[0250] In an eighth aspect, the invention may provide an apparatus for computing a numerical value for uncertainty in the result of a multi-step numerical calculation comprising a sequence of separate calculation instructions defined within a common "basic block", the apparatus configured to implement the following steps comprising:

[0251] (a) identifying a "live-out" variable of the "basic block";

[0252] (b) identifying calculation instructions on who's output the value of the "live-out" variable depends;

[0253] (c) providing a mathematical expression combining the calculation instructions identified at step (b);

[0254] (d) using the mathematical expression of provided at step (c) to compute a numerical value for uncertainty in the "live-out" variable;

wherein the uncertainty value computed at step (d) is the uncertainty in the result of the multi-step numerical calculation.

[0255] The invention includes the combination of the aspects and preferred features described except where such a combination is clearly impermissible or expressly avoided.

SUMMARY OF THE FIGURES

[0256] Embodiments and experiments illustrating the principles of the invention will now be discussed with reference to the accompanying figures in which:

[0257] FIG. **1** schematically shows an apparatus according to embodiments of the invention;

[0258] FIG. **2** shows a schematic representation of approximating distributions for two data sets of measurement data;

[0259] FIG. **3** shows a schematic representation of calculating an arithmetic operation upon the two approximating distributions for two data sets of measurement data of FIG. **2**;

[0260] FIG. **4** shows a schematic representation of calculating an arithmetic operation upon the two approximating distributions for two data sets of measurement data of FIG. **2**;

[0261] FIG. **5** shows a schematic representation of a processor according to an embodiment of the invention;

[0262] FIG. **6**A shows a schematic representation of a processor according to an embodiment of the invention;

[0263] FIG. **6**B shows a schematic representation of a bit-level representation of distributional data according to an embodiment of the invention employing TTR (or RQHR) representation;

[0264] FIG. **6**C shows a schematic representation of a module for conversion of an in-memory array of particle samples (e.g., data/measurement samples) including distributional information/data according to an embodiment of the invention employing TTR (or RQHR) representation;

[0265] FIG. **6**D shows output plots of a distribution (a) of input data, and the result (b) of uncertainty tracking of the input data with autocorrelation tracking on or off, and the ground truth results obtained by exhaustively evaluating x*x for all the samples of x;

[0266] FIG. **6**E shows a schematic representation of a distributional co-ALU which takes as input two distributional source registers and their origin addresses, according to an embodiment of the invention employing TTR (or RQHR) representation;

[0267] FIG. **7** shows steps on a process of generating a SoDD representation of uncertainty illustrating distributional information/data according to an example of the invention;

[0268] FIG. **8** shows a table of origin addresses of data elements used in a register according to an example of the invention;

[0269] FIG. **9** shows a series of RISC-V ISA instructions associated with the table of FIG. **8**;

[0270] FIG. **10** shows two distributions each of which is a representation of uncertainty, and a further distribution which is the result of executing the arithmetic operation of addition on them according to an example of the invention;

[0271] FIG. **11**(*a*)-(*e*) show distributions each of which is a representation of uncertainty, and further distributions which are the result of executing the arithmetic operation of: (b) addition, (c) subtraction, (d) multiplication, (e) division, on them according to an example of the invention;

[0272] FIG. **12**(*a*)-(*e*) show distributions each of which is a representation of uncertainty, and further distributions which are the result of executing the arithmetic operation of: (c) addition, (d) subtraction, (e) multiplication, (f) division, on them according to an example of the invention;

[0273] FIG. **13**A shows distributions representing uncertainty in a calculation of a thermal expansion coefficient calculated by applying arithmetic operations on distributions representing uncertainty in sensor measurement data, according to an example of the invention;

[0274] FIGS. **13**B and **13**C show distributions in measurements made by a temperature sensor (FIG. **13**A), and the resulting measurement uncertainty distributions (FIG. **13**B) representing uncertainty in the sensor measurement data, according to an example of the invention;

[0275] FIGS. **13**D, **13**E and **13**F show distributions of uncertainty in measurements made by a quantum phase estimation (QPE) circuit of a variational quantum eigensolver device implementing a variational quantum eigensolver algorithm, the uncertainty distributions representing uncertainty in the measurement data according to an example of the invention;

[0276] FIG. **14** shows examples [graphs: (a), (b), (c) and (d)] of the probabilities of sets {n} of n values within each one of four different obtained data sets, and shows a sub-set of values from an obtained data set together with probability values for the occurrence of each value within the sub-set.

Tuples formed from data values and associated probability values are shown. An approximating probability distribution is shown comprising a series of Dirac delta functions (SoDD) generated using the tuples;

[0277] FIG. **15** shows a hardware block diagram of one embodiment of the invention;

[0278] FIG. **16** shows an example probabilistic predicate tree;

[0279] FIG. **17** shows a flow chart of a process according to the invention;

[0280] FIGS. **18**(*a*) and **18**(*b*) show a program code (FIG. **18**(*a*)) and an assembly language instruction sequence (FIG. **18**(*b*)) generated by a compiler, from that program code;

[0281] FIG. **19** shows the program code (FIG. **18**(*a*)) and a TAD basic block therefor, together with calculation steps for calculation of uncertainty at stages of the basic block;

[0282] FIG. **20** shows the TAD basic block of FIG. **19** together with calculation steps for calculation of uncertainty at stages of the basic block;

[0283] FIG. **21** shows the TAD basic block of FIG. **19** together with calculation steps for calculation of uncertainty at stages of the basic block;

[0284] FIG. **22** shows the TAD code separated into multiple basic blocks.

DETAILED DESCRIPTION OF THE INVENTION

[0285] Aspects and embodiments of the present invention will now be discussed with reference to the accompanying figures. Further aspects and embodiments will be apparent to those skilled in the art. All documents mentioned in this text are incorporated herein by reference.

EXAMPLE

[0286] FIG. **1** shows an apparatus **1** according to an embodiment of the invention. The apparatus comprises a computing apparatus **3** which is connected in communication with a measurement device **2** in the form of a sensor unit (e.g. an accelerometer, a magnetometer etc.) which is configured to generate measurements of a pre-determined measurable quantity: a measurand (e.g. acceleration, magnetic flux density etc.). The computing apparatus **3** includes a processor unit **4** in communication with a local buffer memory unit **5** and a local main memory unit **6**. The computing apparatus is configured to receive, as input, data from the sensor unit representing measurements of the measurand (e.g. acceleration, magnetic flux density etc.), and to store the received measurements in the local main memory unit **6**. The processor unit **4**, in conjunction with the buffer memory unit **5**, is configured to apply to the stored measurements a data processing algorithm configured for sampling the sensor measurements stored in main memory, so as to generate a sample set of measurements which represents the uncertainty in measurements made by the sensor unit **2** while also representing the measurand as accurately as the sensor unit may allow.

[0287] This representation of uncertainty is achieved by generating the sample set such that the distribution of measurement values within the sample set represent the probability distribution of measurements by the sensor unit **2**. The computing apparatus is configured to store the sample set, once generated, in its main memory **6** and/or to transmit (e.g. via a serial I/O interface) the sample set to an external memory **7** arranged in communication with the computing apparatus, and/or to transmit via a transmitter unit **8** one or more signals **9** conveying the sample set to a remote receiver (not shown). The signal may be transmitted (e.g. via a serial I/O interface) wirelessly, fibre-optically or via other transmission means as would be readily apparent to the skilled person.

[0288] In this way, the computing apparatus is configured to generate and store any number (plurality) of sample sets, over a period of time, in which the distribution of measurement values within each sample set represents the probability distribution of measurements by the sensor unit **2**. Similarly, the computing apparatus is configured to generate and store any number (plurality) of sample sets, as generated by different modes of operation of the sensor unit **2** or as generated by different sensor unit **2**. In other words, a first sample set stored by the computing apparatus may be associated with a first sensor unit and a second sample set stored by the computing apparatus may be associated with a second sensor unit which is not the same as the first sensor unit. For example, the first sensor unit may be a voltage sensor (electrical voltage) and the second sensor unit may be a current sensor (electrical current). In both cases, the computing apparatus may store distributions of measurement values made by each sensor, respectively, which each represent the probability distribution of measurements by that sensor unit **2**.

[0289] The computing apparatus **3** is configured to implement a method for the encoding and computation on the distributions of data stored in the main memory unit, as follows.

[0290] As a first step, the computing apparatus **3** obtains a first set of measurement data items from the sensor unit **2** and obtains a second set of measurement data items from the sensor unit **2** (which may be the same sensor or a different sensor). These data sets are obtained from storage in the main memory unit and are stored in the buffer memory unit **5** for the duration of processing upon them. The processor unit **4** then applies to the first and second sets of measurement data items a process by which to generate an approximate distribution of the respective set of measurement data items. This process may be any one of the processes described above for generating an N-dimensional SoDD-based representation of fixed type, referred to above as FDR i.e., RQHR, PQHR, MQHR or TTR, or N-dimensional CMR representation. The processor unit applies the same process to each of the first and second measurement data sets, separately, so as to produce a first N-tuple containing parameters encoding a probability distribution characterising the distribution of the measurement data items of the first set, and a second N-tuple containing parameters encoding a probability distribution characterising the distribution of the measurement data items of the second set.

[0291] Note that the same process is applied, by the processor unit, the parameters used to encode the distribution of the data items of the second set (i.e. the positions, x_i, of Dirac-δ functions representing data of the second set, and their heights/probabilities, p_i) are the same as the parameters used to encode the distribution of the data items of the first set (i.e. the positions, x_i, of Dirac-δ functions representing data of the first set, and their heights/probabilities, p_i). Of course, the parameters (position, probability) are the same, but not their values.

[0292] FIG. **2** shows a schematic example of this process for the case when the N-dimensional SoDD-based representation is the MQHR representation. The result of applying the MQHR processing to the first measurement data set is generate a first N-dimensional SoDD-based representation **11** of the first measurement data set, which is entirely defined by the parameters of a first N-tuple **10**:

$$\mathcal{R}^{N}_{MQHR}(X) := \left(x_1, \ldots, x_{N_{dd}}, p_1, \ldots, p_{N_{dd}}\right)$$

[0293] Similarly, the result of applying the MQHR processing to the second measurement data set is generate a second N-dimensional SoDD-based representation **13** of the second measurement data set, which is entirely defined by the parameters of a second N-tuple **12**:

$$\mathcal{R}^{N}_{MQHR}(Y) := (y_1, \ldots, y_{Ndd}, p, \ldots, p_{Ndd})$$

[0294] Note that the values of the parameters in the first N-tuple **10** will generally not be the same values as the values of the parameters of the second N-tuple **11**. This is, of course, simply because the distributions of the measurement values in the first and second measurement data sets will generally not be the same as each other.

[0295] The processor unit **4** may be configured to store the first and second N-tuples (**10**, **11**) in the main memory unit **6** for later use in applying an arithmetic operation (propagation) to them. The processor unit may then simply obtain the first and second N-tuples (**10**, **11**) from the main memory unit **6** and place them, for example, in the buffer memory unit **5** for applying an arithmetic operation (propagation) to them as, and when required. In alternative embodiments, the processor may be configured to obtain the first tuple and the second tuple as output from a remote memory unit **7** or by receiving them as output from a receiver **8** in receipt of a signal **9** conveying the first and second tuples from a remote transmitter/source (not shown).

[0296] Next, the processor unit **4** generates a third tuple by applying an arithmetic operation on the first and second N-dimensional SoDD-based representations (**11**, **13**) of the first and second measurement data sets, each of which is entirely defined by the parameters of its respective N-tuple (**10**, **12**):

$$\mathcal{R}^{N}_{MQHR}(X) := \left(x_1, \ldots, x_{N_{dd}}, p_1, \ldots, p_{N_{dd}}\right)$$

[0297] To perform this arithmetic operation, the processor uses parameters (i.e. the positions, x_i, of Dirac-δ functions representing data, and their heights/probabilities, p_i) contained within the first tuple and the parameters (i.e. the positions, x_i, of Dirac-δ functions representing data, and their heights/probabilities, p_i) contained within the second tuple. The result is a third tuple containing parameters encoding a probability distribution representing the result of applying the arithmetic operation on the first probability distribution and the second probability distribution. The processor then outputs the third tuple either to the local memory unit **6**, for storage, and/or to the transmitter unit **8** for transmission **9**.

[0298] The processor unit is configured to implement an arithmetic operation comprising one or more of: addition; subtraction; multiplication; division, or any bivariate operation, e.g., exponentiation and many others. As an example of an arithmetic operation, FIG. **3** schematically represents the addition and multiplication operations on distributions approximated by the first and second N-dimensional SoDD-based representations (**11**, **13**) of the first and second measurement data sets:

$$\tilde{f}^{FDR}_{X}(x) = \sum_{n=1}^{N_{dd}} p_n \delta(x - x_n) \text{ and } \tilde{f}^{FDR}_{Y}(y) = \sum_{n=1}^{N_{dd}} q_n \delta(y - y_n).$$

[0299] These two distributions (where 'FDR'=MQHR in this example) are added together or multiplied together to produce a third N-dimensional SoDD-based representation **15**:

$$\tilde{f}_{\Phi(X,Y)}(z) = \sum_{n=1}^{N_{dd}} \sum_{m=1}^{N_{dd}} p_n q_m \delta(z - \Phi(x_n \cdot y_m)).$$

[0300] The third SoDD-based representation **15** represents the result of addition when the quantity $\Phi(x_n, y_m)$ within this representation is calculated as $\Phi(x_n, y_m)=\Phi^{+}(x_n, y_m)=x_n+y_m$, as described above.

[0301] The third N-dimensional SoDD-based representation **15** represents the result of multiplication when the quantity $\Phi(x_n, y_m)$ within this representation is calculated as $\Phi(x_n, y_m)=\Phi^{\times}(x_n, y_m)=x_n \times y_m$, as described above.

[0302] However, the processor unit is not required to reproduce any of the first, second or third SoDD-based representation (**11**, **13**, **15**) in order to generate the third tuple **15**. This is because, the arithmetic process allows each parameter of the third tuple to be generated using the parameters of the first tuple and the second tuple. Once the parameters of the third tuple are calculated, then this fully encodes the third SoDD-based representation **15**, permitting that third distribution to be reproduced as and when required, and permitting the third tuple to be stored (in local memory unit **5**, or remote memory **7**) and/or transmitted as a signal **9** from the transmitter unit **8**, in a very efficient form.

[0303] This storage/transmission simply requires the parameters (i.e. the positions, z_i, of Dirac-δ functions representing data of the third set, and their heights/probabilities, p_i) encoding a probability distribution characterising the distribution of the data items of the third set **15** of data items that are the result of the arithmetic operation. The parameters used to encode the distribution of the data items of the third set are the same as the parameters used to encode the distribution of the data items of the first set and second set, since the third set is encoded using the same SoDD-based representation (i.e. MQHR in this example) as the first set and second set. To achieve the same format of representation (or same parameters) for the propagated variable Z, the third tuple (**18** of FIG. **4**) is preferably reduced in size to a "compacted" third tuple (**18**B of FIG. **4**) which is an N-tuple. This permits further computation that may involve Z without the representation sizes growing indefinitely. This reduction to a standard size (i.e., N-tuple) also assists in the micro-architectural implementation discussed herein. FIG. **4** sche-

matically shows the process of multiplication applied to two distributions (**11**, **13**) in the MQHR representation, resulting in a third distribution **15** also in the MQHR representation. FIG. **4** also shows the implementation this operation in terms of using the parameters (position, probability) of the N-tuple for the first distribution **11** of the two distributions, and using the parameters (position, probability) of the N-tuple for the second distribution **12** of the two distributions, to generate a third tuple **18** parameters comprising parameters (position, probability) having values defining the distribution resulting from the multiplication. This third tuple is then preferably used to generate a "compacted" third tuple **18**B, as described above, which has the same size (an N-tuple) as that of the first and second tuples (**10**, **12**).

[0304] In particular, the positions, z_k, of Dirac-δ functions representing data of the third set **15** are given by the values of $\Phi(x_n, y_m)$. The heights/probabilities, p_k, of the Dirac-δ functions representing data of the third set are given by the product, $p_k=p_n\times p_m$, of the heights/probabilities of the Dirac-δ functions representing data of the first (p_n) and second (p_m) data sets. This may be summarised, schematically, in the case of a multiplication of the first and second distributions, as:

$$(x_1, \ldots, x_{N_{dd}}, p_1, \ldots, p_{N_{dd}}) \times (y_1, \ldots, y_{N_{dd}}, p_1, \ldots, p_{N_{dd}}) = (z_1, \ldots, z\textcircled{?}, p_1, \ldots, p\textcircled{?})$$

? indicates text missing or illegible when filed

[0305] The invention is not limited to SoDD-based representations (such as MQHR), and the first, second and third tuples may generally contain parameters, according to other representations, encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the first, second and third sets such as described in any of the examples discussed above. The first, second and third tuples may contain parameters, according to other representations, encoding the probability of data items within the probability distribution characterising the distribution of the data items of the first set. For example, as discussed above, the first, second and third tuples may contain parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the first set.

[0306] The first and second tuples are each an N-tuple in which N>1 is an integer, and the third tuple is an M-tuple for which $N^2/2<M<2N^2$ in which N>1 is an integer. The compacted third tuple, if generated, is preferably an N-tuple. Efficiencies of data storage and transmission are greatly enhanced by the invention by using the first and second tuples to represent data distributions, but also by using the third tuple to represent the third distribution of data. Much less data are required to represent the data distributions, and this greatly lowers the burden on memory space for storing the data distributions according to the invention. Furthermore, an efficient means of performing arithmetic operations on the data sets is provided which greatly reduces the computational burden on a computer system.

[0307] As an example, to better illustrate a use of the invention in this context, the first and second sets of data may typically comprise samples of a respective first and second random measurement variable, representing uncertainty in the measurements. As an example, the first data set may comprise measurements from a voltage sensor, e.g. of a voltage (V) across a circuit component, and the second data set may comprise measurements from a current sensor, e.g. of a current (I) through the circuit component. Consequently, the third set of data, generated by applying the arithmetic operation of multiplication to the distributions of the first and second sets of data, may represent the uncertainty in the electrical power (P=I×V) dissipated by the circuit component. In this way, the monitoring of the power dissipation of the circuit component, and a representation of the uncertainty in measured power, becomes not only possible but also much more efficient in terms of memory requirements of the monitoring computer and in terms of the processing/computing burden on that computer.

EXAMPLE

Microarchitecture

[0308] The disclosures above provide efficient binary number representations of uncertainty for data whose values are uncertain. The following describes an example hardware architecture for efficiently performing computation on these representations. For example, the example hardware architecture may be an example of the processor unit **4** described above with reference to FIG. **1**. The processor unit **4** may be provided in the form of a microarchitecture described in more detail below, with reference to FIG. **5** and FIGS. **6**A, **6**B and **6**C.

[0309] The above binary number representations (tuples) for data effectively implement a trade-off between the number of bits used in representations and the accuracy of a representation of empirical probability distributions: in other words, fewer bits means greater efficiency of uncertainty representation, but at the cost of some acceptable loss of accuracy in that representation. The examples of uncertainty representations presented above allow computing systems in general to efficiently represent uncertain quantities even when those quantities have rarely occurring high-moment outliers.

[0310] The uncertainty representations and algorithms for performing arithmetic on them, as presented above, may be implemented in a microarchitecture for computing on data associated with distributions (distributional information/data). An instruction set architecture (ISA) may be used with the microarchitecture which is an extension of the RISC-V 32-bit ISA. The microarchitecture may execute existing RISC-V programs unmodified and whose ISA may be extended to expose new facilities for setting and reading distributional information/data without changing program semantics. The microarchitecture may represent and propagate distributional information/data and provide uncertainty-awareness at the software level.

[0311] Uncertain data are ubiquitous in computing systems. One common example is sensor measurements, where the very nature of physical measurements means there is always some degree of uncertainty between the recorded value (the measurement) and the quantity being measured (the measurand). This form of measurement uncertainty is often quantified by performing repeated measurements with the measurand nominally fixed and observing the variation across measurements using statistical analysis or noting the number of significant digits. Such numerically quantified

uncertainty is referred to in the literature as aleatoric uncertainty. Uncertainty may also exist when there is insufficient information about a quantity of interest. For example, the training process for neural networks determines values of per-neuron weights, which training updates across training epochs by backpropagation. Because of the large space of possible values for the parameters that control the training process such as the momentum for gradient updates and step size (so-called hyperparameters), weights in a neural network model are initially uncertain but eventually converge on a narrower distribution of weight values as a result of the training process. In the case of a sensing device, the random errors that can occur in the measurement are considered an aleatoric uncertainty. Epistemic uncertainty refers to our imperfect knowledge or ignorance of parameters of the examined phenomenon. The present invention, as implemented by the microarchitecture or otherwise, may encode, represent and propagate distributional information/data (e.g. probability distributions, frequency distributions etc.) representing uncertainty in measurement data made by a measurement apparatus (e.g. sensor) and/or may propagate distributional information/data defining distributional weights of an artificial neural network.

[0312] Such uncertainty in values, resulting from incomplete information on the values they should take, is often referred to in the research literature as epistemic uncertainty. Despite the increasing relevance of both of these types of uncertain data in modern computing systems, modern computer architectures neither have support for efficiently representing epistemic uncertainty, nor for representing aleatoric uncertainty, let alone arithmetic and control-flow on such values. Computer architectures today represent both kinds of uncertain values with single point values or "particle" values (i.e., data with no associated distribution), usually by taking the mean value as the representation for use in computation. Hereafter, for conciseness, we refer to single point values (i.e., data with no associated distribution) as "particle" values. The microarchitecture disclosed herein provides a non-intrusive architectural extension to the RISC-V ISA, for computing with both epistemic as well as aleatoric uncertainty.

[0313] The microarchitecture may allow programs to associate distributional information with all of the floating-point registers and/or integer registers and by extension all chosen memory words that are used to load and store register values. Arithmetic instructions may propagate the distributional information/data associated with the registers that are the source operands to the destination register and back to memory on a store operation.

[0314] Examples of representations are given above (e.g. SoDD-based representations, CMR representations) for efficiently capturing the uncertainty of each memory element inside the microarchitecture by means of discrete probability distributions. In this way, computer representations for variables with distributional information can be seen as analogous to computer representations for real-valued numbers such as fixed-point floating-point representations. In particular, fixed-point representations use a fixed number of bits for whole and fractional parts of a real-valued quantity and represent approximate real values as fixed spacings over their dynamic range while floating-point representations represent real-valued quantities with an exponential expression that permits representing a wider dynamic range but results in non-uniform spacings of values on the real number line. Just as fixed-point and floating-point number representations trade the number of bits in their representations of real-valued quantities for accuracy with respect to the represented real-valued number, the distribution representations/data disclosed herein trade the bits in representations for accuracy of representing empirical probability distributions.

[0315] The microarchitecture may both represent and track uncertainty across a computation transparently to the default semantics of the default RISC-V ISA. The microarchitecture may permit a user to make probabilistic and statistical queries on the uncertainty at any point in the computation.

[0316] Because the examples of representations disclosed herein (e.g. SoDD-based representations, CMR representations) are of finite length, they are an approximation of the probability density function of the original discrete random variable. To recap, let $\delta(x)$ be the Dirac distribution centred on x. Given a "particle" value $x_0 \in \mathbb{R}$, we define the particle value as the distribution $\delta(x-x_0)$. Using this definition, we represent an array of M particle values as a sum of weighted Dirac deltas, in the form

$$f_X(x) = \sum_{n=1}^{M} p_n \delta(x - x_n)$$

where $p_n \in [0,1]$ with:

$$\sum_{n=1}^{M} p_n = 1$$

[0317] In the case of M particle values we set p_n equal to the probability of appearance of each of the M values, which is 1/M. Thus, the updated Dirac deltas distribution of the M particle values takes the form:

$$f_X(x) = \frac{1}{M} \sum_{n=1}^{M} \delta(x - x_n)$$

[0318] This distribution is an accurate representation, which we will refer to as the Dirac mixture representation. The Dirac mixture representation may be used as a reference distributional representation of particle data, in order to define the following three compact distribution representations disclosed in more detail above. We define a distribution function of a random variable X, by the function:

$$F(x) = \mathbb{P}(X \leq x).$$

[0319] For discrete random variables, i.e., variables that take values only in some countable subset of real numbers $\mathbb{R}$, we define the probability mass function of a discrete random variable X as:

$$f(x) = \mathbb{P}(X = x).$$

[0320] Here $\mathbb{P}(X=x)$ is the probability that the random variable X has the observed value x. We define the mean value, or expected value, of a random variable X with "probability mass" function $f(x)$ as:

$$\mathbb{E}(X) = \sum_{x:\, f(x)>0} x f(x).$$

Centralized Moments Representation (CMR):

[0321] This representation uses the expected value and the first N centralized moments of the Dirac mixture representation of an array of particle data. We exclude the first centralized moment which is always equal to zero. We define the N-th order centralized moment representation (CMR) of a discrete random variable X as the ordered N-tuple defined above. For completeness, we define the k-th moment, m_k, (k=positive integer) of the random variable X as the expectation value:

$$m_x = \mathbb{E}(X^k).$$

[0322] We define the k-th centralized moment, σ_k, of random variable X as the expectation value:

$$\sigma_k = \mathbb{E}\left((X - m_1)^k\right).$$

[0323] For example, σ_2 is the variance of the random variable. The ordered N-tuple is then:

$$R^N_{CMR}(X) := (\mathbb{E}(X), \sigma_2(X), \ldots, \sigma_N(X)).$$

Regularly-Quantized Histogram Representation (RQHR):

[0324] In this representation we may divide the range of the input particle values into N bins of equal range size L. We define the height of each bin $1 \le i \le N$ as p_i and set it equal to the relative frequency of the input particle data falling within the interval of i-th bin. This results in a set of regularly positioned histograms, which can be modelled as a Dirac mixture, where each Dirac delta lies in the middle of the interval of each histogram with probability mass equal to p_i. Since the range size L is constant, by knowledge of the position x_0 of the first Dirac delta, it is possible to calculate the position of the rest of them. Consequently, we may define the regularly quantized histogram representation (RQHR) of X consisting of N Dirac deltas as the ordered (N+2)-tuple:

$$R^N_{RQHR}(X) := (0, L, p_1, \ldots, p_N).$$

Telescoping Torques Representation (TTR):

[0325] The telescoping torques representation (TTR) recursively constructs a new set of N Dirac deltas from a Dirac mixture with any number of elements in $\log_2 N$ steps. At each step the construction divides the given Dirac mixture into two: those that lie below the mean value of the mixture and those that lie above the mean value of the mixture, to obtain twice the number of Dirac mixtures. FIG. **7** shows an example for N=4 and starting from a Dirac mixture of 8 "particles". The mean value of the initial Dirac mixture is equal to μ_0=22.5. At the step 0 of the TTR construction, we define the 0th-order TTR of the given mixture as a Dirac delta with probability mass equal to the sum of the probability masses of all Dirac deltas in the mixture (which is 1.0) and sitting at the mean value μ_0. At step 1, we consider the two mixtures that lie, respectively, below μ_0 and above μ_0 (each mixture consisting of four Dirac deltas in the case of this example) and repeat the process for each mixture. In this way one defines the 1st-order TTR of the initial mixture as the union of the 0th-order TTRs of each sub-mixture. In the case of the given example, this corresponds to two Dirac deltas sitting at the mean values of the sub-mixtures and both with "probability mass" equal to 0.5. At step 2, the process repeats, further dividing the sub-mixtures of step 1 into two and finding their 0th-order TTRs. This provides a TTR of order n=2 for the initial mixture with $N=2^n=4$ Dirac deltas. If $x_1, \ldots, x_N$ are the positions of the Dirac deltas in the nth stage of the construction, where $n=\log_2 N$ and if $p_1, \ldots, p_N$ are their "probability masses", then the nth-order TTR of X as the ordered 2N-tuple:

$$R^N_{TTR}(X) := (x_1, \ldots, x_N, p_1, \ldots, p_N).$$

[0326] Both TTR and RQHR representations are of fixed size and do not increase in size as computation progresses.

Arithmetic Operations on Representations

[0327] Given two discrete random variables X and Y with probability mass functions represented using any of the representations disclosed herein, the algorithms disclosed herein may be implemented in order to compute the representation of the "probability mass" function of the discrete random variable $Z=\Phi(X; Y)$, for $\Phi(X; Y)$ as one of addition, multiplication, subtraction and division. The microarchitecture **4** supports these arithmetic operations for all distribution representations disclosed herein. The microarchitecture, and the invention in general, may calculate more complex arithmetic operations e.g., fused multiplication and addition or square root, by expressing them in terms of the aforementioned basic arithmetic operations, as would be readily apparent to the skilled person.

[0328] The RQHR and TTR representations (and other SoDD-based representations), for example, are both in the form of a series of Dirac deltas (i.e. "SoDD"), which may be represented as a vector, if desired, of given position and probability mass. Assuming two random variables X and Y, their addition and multiplication results from the circular convolution of Dirac deltas positions and probability mass vectors. For the following parts of this disclosure, we will

focus on TTR, but the same principles apply to RQHR (and other SoDD-based representations).

[0329] Algorithm 1 shown in Table 2 provides an example of an algorithm implemented by the microarchitecture of the present invention for addition of two input discrete random variables represented using TTR of size N_{TTR}. The results of the operations on the Dirac delta positions and masses of the two input variables are temporarily stored in the variable "destVar", which is of Dirac mixture representation type of size $N_{TTR} \times N_{TTR}$. After the completion of calculations, the microarchitecture converts the "$destVar_{DM}$" to "$destVar_{TTR}$", using the algorithm illustrated in Table 2.

TABLE 2

Algorithm 1: Addition of uncertainty representations based on Dirac deltas.

```
1  Inputs : srcVar1_TTR, srcVar2_TTR
2  Function uncertainAdd_TTR(srcVar1_TTR, srcVar2_TTR)
3  |  destVar_DM ← new DMvar(N_TTR × N_TTR);
4  |  for i ← 1 to N_TTR do
5  |  |  for j ← 1 to N_TTR do
6  |  |  |  index ← (i − 1) * N_TTR + j;
7  |  |  |  destVar_DM.position[index] ←
   |  |  |     srcVar1_TTR.position[i] + srcVar2_TTR.position[j];
8  |  |  |  destVar_DM.pmass[index] ←
   |  |  |     srcVar1_TTR.pmass[i] * srcVar2_TTR.pmass[j];
   |  |  end
   |  end
9  |  destVar_TTR ← convertDMToTTR(destVar_DM);
10 |  return destVar_TTR
```

[0330] A similar process is required for the multiplication of two variables in TTR, as shown in Table 3.

TABLE 3

Algorithm 2: Multiplication of uncertainty representations based on Dirac deltas.

```
1  Inputs : srcVar1_TTR, srcVar2_TTR
2  Function uncertainAdd_TTR(srcVar1_TTR, srcVar2_TTR)
3  |  destVar_DM ← new DMvar(N_TTR × N_TTR);
4  |  for i ← 1 to N_TTR do
5  |  |     for j ← 1 to N_TTR do
6  |  |  |     index ← (i − 1) * N_TTR + j;
7  |  |  |     destVar_DM.position[index] ←
   |  |  |        srcVar1_TTR.position[i] * srcVar2_TTR.position[j];
8  |  |  |     destVar_DM.pmass[index] ←
   |  |  |        srcVar1_TTR.pmass[i] * srcVar2_TTR.pmass[j];
   |  |     end
   |  end
9  |  destVar_TTR ← convertDMToTTR(destVar_DM);
10 |  return destVar_TTR
```

[0331] The difference between these two algorithms is that the positions of the input Dirac deltas are multiplied in the case of Algorithm 2 instead of being added as in the case of Algorithm 1.

[0332] Algorithms 1 and 2, showcase the merits of the Dirac delta representations. The required operations for arithmetic propagation require element-wise operations, which are highly parallel and their calculation can be optimized. Moreover, the result of an arithmetic operation on two variables is also in a Dirac delta mixture form, which can be converted to the intended representations using procedures that apply for particle data and no extra hardware or software logic.

[0333] An addition or multiplication of a particle value to a variable in TTR form, results in an offset or scaling of the position of the Dirac deltas of the representation. With that in mind, the subtraction of two uncertain variables in TTR is achieved by negating the TTR of the subtrahend using multiplication with −1. For the case of division of TTR variables, we define it as a multiplication using the reciprocal of the divisor variable. The reciprocal of a TTR variable, can be constructed by calculating the reciprocals of the input Dirac deltas positions.

Bayes-Laplace Rule

[0334] Let e be a random variable that takes on instance values θ and which has probability mass function $f_\Theta(\theta)$. Let $f_X(x|\theta)$ be the probability mass function of the distribution of the random variable X, given a parameter Θ, i.e., $Pr\{X=x|\theta\}$. The parameter Θ is typically a variable in machine state such as a word in memory corresponding to a weight in a neural network model being executed over the microarchitecture. In Bayesian neural networks, there is epistemic uncertainty about such weights and it is the goal of the training procedure to estimate their (posterior) distribution based on a combination of prior knowledge and values seen during training. The Bayes-Laplace rule gives us the expression for the probability mass function of the random variable Θ given one or more "evidence" samples x of the random variable X. Then, given a vector of N samples of the random variable X, i.e., given $x=\{x_1, x_2, \ldots x_N\}$, $f_\Theta(\theta|x)$ is:

$$f_\Theta(\theta \mid x) = \frac{f_X(x \mid \theta) f_\Theta(\theta)}{\sum_{\theta \in \mathcal{D}(\Theta)} f_X(x \mid \theta) f_\Theta(\theta)}.$$

[0335] The left-hand side of this equation is often referred to as the "posterior distribution" of the parameter Θ. The probability mass function $f_\Theta(\theta)$ is referred to as the "prior distribution" for the parameter Θ and the "likelihood" is computed as:

$$f_X(x \mid \theta) = \prod_{\{x_1, x_2, \ldots, x_N\}} f_X(x \mid \theta).$$

[0336] The likelihood is often referred to as the sampling distribution. The Bayes-Laplace rule for computing the posterior distribution is an invaluable operation in updating the epistemic uncertainty of program state. In contemporary systems, this update is widely considered to be computationally challenging because of the need to perform the integral or equivalent summation in the denominator of the above equation for $f_\Theta(\theta|x)$. The present invention permits computation of the posterior distribution using the representations of uncertainty (e.g. SoDD representations) of the prior distribution, the sampling distribution and the set of "evidence" samples.

[0337] Examples of a microarchitecture for the processor unit **4** for computation on distributions of data, are shown in FIG. **5** and FIGS. **6**A, **6**B and **6**C. In each example, the microarchitecture comprises microarchitecture unit **20** comprising a floating-point register file **28** configured to contain floating-point data. The floating-point register file contains a first register file **28**A configured for containing particle data items, and a second register file **28**B configured for con-

taining distribution data. The distributional data represents distributions (e.g. SoDD-based representations) that are uncertainty representations associated with respective particle data items in the first register file **28**A. The floating-point register file **28** associates a given particle data item in the first register file **28**A with its associated distributional data within the second register file **28**B by assigning one common register file entry identifier (fi) to both a given particle data value within the first register file **28**A and the distributional data entry in the second register file **28**B that is to be associated with the particle data in question. For example, the first register entry in both the first register file **28**A and the second register file **28**B are identified by the same one register entry identifier "f0". Similarly, the last register entry in both the first register file **28**A and the second register file **28**B are identified by the same one register entry identifier "f31". Of course, the same applies to intermediate register entry identifiers, such that the i^{th} register entry in both the first register file **28**A and the second register file **28**B are identified by the same one register entry identifier "fi".

[0338] A first arithmetic logic unit **25** is configured for executing arithmetic on particle data items selected from the first register file **28**A, and a second arithmetic logic unit **26** is configured for executing arithmetic on distribution data selected from the second register file **28**B. The microarchitecture **20** of the processor **4** is configured to implement the following steps. The first arithmetic logic unit **28**A executes an arithmetic operation (e.g. addition, subtraction, multiplication or division) on two floating-point particle data items selected from the first register file **28**A, and outputs the result. Simultaneously, the second arithmetic logic unit **28**B executes the same arithmetic operation on two items of distribution data representing distributions selected from the second register file **28**B that are associated with the data items that were selected from the first register file **28**A, and outputs the result. Notably, the arithmetic operation executed on the distribution data selected from the second register file **28**B is the same as the arithmetic operation executed on the data items selected from the first register file **28**A. As a result, the output of the second arithmetic logic unit **26** is further distributional data representing uncertainty associated with result of the arithmetic operation (e.g. addition, subtraction, multiplication or division) executed on the data items selected from the first register file **28**A.

[0339] As an example, consider a case in which the first arithmetic logic unit **25** selects particle data items from within the first register file **28**A at register file locations/entries: f1 and f2, for adding together, and outputting the result to register file location/entry f0 within the first register file **28**A (e.g. for subsequent output from the processor, or for use in further arithmetic operations). The selecting of particle data items from within the first register file **28**A at register file locations/entries: f1 and f2, triggers concurrent selection by the second arithmetic logic unit **26** of distributional data items from within the second register file **28**B at register file locations/entries: f1 and f2, for adding together, and outputting the result to register file location/entry f within the second register file **28**A (e.g. for subsequent output from the processor, or for use in further arithmetic operations). This can be summarised as follows, using the naming convention for the floating-point registers of RISC-V architecture, let f0 be a floating-point register of the microarchitecture:

*f*0:particle=*f*1:particle+*f*2:particle Arithmetic operation:

*f*0:distribution=*f*1:distribution+*f*2:distribution Arithmetic operation:

[0340] The outputting of the results of this arithmetic operation, by the microarchitecture unit **20** may comprise one or more of: storing the output in a memory; transmitting a signal conveying the output (e.g. electronically to another circuit component, or to a memory, or wirelessly to a remote receiver). The first register file **28**A and the second register file **28**B are configured to contain at least a first particle data item and associated distributional data (e.g. at f1) and a second particle data item and associated distributional data (e.g. at f2), but may contain many more data items (e.g. up to 32 in this example: f0 to f31).

[0341] The second register contains the distribution data (e.g. f1: distribution, f2: distribution) associated with a given particle data item in the form of a tuple, e.g. of any type disclosed herein, containing parameters encoding a probability distribution characterising the uncertainty representation associated with the particle data item in question. The parameters used to encode the distribution data of all of the distributional data items of the second register file **28**B are the same as each other, at least during a given arithmetic operation. Consequently the result of the arithmetic operation applied to the two selected tuples of distributional data, is a third tuple (f0: distribution) defined using parameters contained within the first tuple, f1: distribution, and using parameters contained within the second tuple, f1: distribution. The third tuple, f0: distribution, contains parameters encoding new further distribution data which is stored at location f0 in the second register file **28**A.

[0342] The processor **4** also comprises a microarchitecture unit (**21**, FIG. **5**; **21**B, FIG. **6**) comprising an integer arithmetic logic unit **27** arranged to implement arithmetic operations on integer particle values contained within an integer register file (**29**, FIG. **5**; **29**A & **29**B, FIG. **6**). In the example of FIG. **5**, only integer particle values are contained in the integer register file **29**, and the integer arithmetic logic unit **27** is arranged to implement arithmetic operations only on integer particle values contained within an integer register file **29**. However, in the example of FIG. **6**A, in the microarchitecture **21**B integer particle values are contained in a first register file **29**A and associated distributional values are contained in a second distributional register file **29**B. Furthermore, the microarchitecture **21**B comprises an execution unit **24** including not only a first arithmetic logic unit **27** (for integer operations) shown in the example of FIG. **5**, but also containing a second arithmetic logic unit **31** configured to perform arithmetic operations on distributional information associated with integer particle data. In this way, the microarchitecture **21**B is arranged to implement arithmetic operations only on integer particle values and their associated distributional data contained within the second register file **29**B. The operation and functioning of the first and second arithmetic logic units (**27**, **31**) interact with the first and second register files (**29**A, **29**B) such that arithmetic operations performed on integer particle values are also concurrently performed on associated distributional values in the same manner as described above with reference to the floating-point registers and arithmetic units. In summary, the above relation:

*f*0:particle=*f*1:particle+*f*2:particle Arithmetic operation:

*f*0:distribution=*f*1:distribution+*f*2:distribution Arithmetic operation:

[0343] applies equally to integer data arithmetic operations.

[0344] In this way, the floating-point and/or integer register files (**28**A, **28**B, **29**A, **29**B) in the microarchitecture may associate all floating-point and/or integer registers with distributional information. When an instruction reads from a floating-point and/or integer register which has no distributional information, the behaviour is unchanged from a conventional architecture. The semantics in the presence of distributional information is to return the mean. The number of floating-point registers and their conventional particle part remains unchanged.

[0345] In summary, FIG. **5** shows a microarchitecture example which does not associate integer registers with uncertainty and thus the integer register file is of conventional design, whereas FIG. **6** shows a microarchitecture example which does associate integer registers with uncertainty and thus the integer register file is according to the invention. Both figures illustrate an example of a processor **4** according to an embodiment of the invention (e.g. the processor unit of FIG. **1**). The processor unit, in these examples, is configured to implement an RISC-V ISA. Accordingly, the processor unit comprises a RISC-V Instruction Fetch Unit **22** arranged in communication with an RISC-V Decode Unit **23** of the processor unit. The RISC-V Instruction Fetch Unit **22** is configured to fetch instructions from the computer **3** (FIG. **1**) and holds each instruction as it is executed by the processor unit **4**. The Fetch Unit issues instructions to the Decode unit **23** which, in turn, is responsive to a received Fetch instruction so as to decode the received instruction and issue a consequent instruction to a RISC-V Execution Unit **24** of the processor unit, to execute the decoded instruction. Upon receipt of an instruction, the RISC-V Execution Unit **24** issues instructions to the arithmetic logic units (ALU) of the processor unit, to execute an arithmetic operation on floating-point particle data **28**A and their associated distributional information/data **28**B, according to the invention, and optionally also to execute an arithmetic operation on integer particle data **29**A.

[0346] In this way, the processor preferably comprises extended register files (**28**A, **28**B; **29**A, **29**B) according to the invention, comprising a first register file (**28**A, **29**A) that can store floating-point data, or integer data, and a second register file (**28**B, **29**B) that can store distributional information/data. This extended floating-point register file associates all floating-point (or integer) registers within the first register file (**28**A, **29**A), with distributional information within the second register file (**28**B, **29**B). The execution unit **24** follows the algorithms disclosed herein to cause an extended functional unit (**20**, **21**B) containing a floating-point (or integer) distribution arithmetic unit (**26**, **31**) to execute arithmetic operations on distributional data (second register file **28**B, **29**B) associated with floating-point (or integer) particle data (first register file **28**A, **29**A) within a floating-point (or integer) register files **28**A or **29**A.

[0347] Details of the microarchitecture illustrated in FIGS. **6**A, **6**B and **6**C

[0348] FIG. **6**B shows the bit-level representation of an RQHR or TTR of size N. Nomenclature and symbology in FIG. **6**B is as follows:

N:	Size of the distributional representation. It contains N support positions and N probability masses, each 64 bits long
$n_{samples}$:	The 64-bit long number of particle samples that Laplace used to generate the distributional representation
U:	Beginning of the Nth support position bitfield. Starts after (N − 1) positions of 64 bits plus 64 bits for $n_{samples}$ U = 64N
V:	End of the Nth support position bitfield. V = U + 63 = 64 (N + 1) − 1 bits
W:	Beginning of the 1^{st} probability mass bitfield. W = V + 1 = 64(N + 1) bits
X:	End of the 1^{st} probability mass bitfield. X = W + 63 = 64(N + 2) − 1 bits
Y:	Beginning of the Nth probability matt bitfield. Starts after N positions and (N − 1) masses plus $n_{samples}$ Y = 2N*64 bits
Z:	End of the Nth probability mass bitfield. Z = Y + 63 = 64(2N + 1) − 1 bits

[0349] The lower-order 64 bits store the number of particle samples used to derive the distributional representation. The next N 64-bit values store the support positions of the representation. They are followed by the N 64-bit values of the probability masses. Let f0 be a conventional floating-point register of the microarchitecture (e.g., RISC-V RV32IMFD) and let df0 be the respective microarchitecture-level distributional floating-point register. The microarchitecture according to preferred embodiments performs all arithmetic and logic instructions on both the conventional and distributional registers in parallel. For example, the addition of source registers f1 and f2 into destination register f0 also triggers the addition of the distributional information of registers df1 and df2 into the distributional register df0. The semantics of non-distributional register values and operations on them remain unchanged. Tracking of distributional information happens in parallel to (and not affecting the behaviour of) the non-distributional architectural state. The microarchitecture according to preferred embodiments extends both integer and floating-point arithmetic and logic units (ALUs) with two distributional co-ALUs. The conventional, unchanged ALU operates on the particle values of the source registers. The distributional co-ALU performs the same operation on the distributional representations of the source registers using the algorithms of Table 2 and Table 3 noted above. The distributional co-ALU may be configured to calculate statistics of the distributional representation of a source register. Examples include the statistical measures described herein (e.g., above), such as, though not limited to the following.

[0350] The N^{th} centralized moment:

[0351] For any integer N>0, the N^{th} moment of random variable X is given by $\sigma_N = E((X - E(X))^N)$ where:

$$E(X) = \sum_{i=0}^{K} d_{pos}[i] d_{mass}[i]$$

Here, $d_{pos}[i]$ and $d_{mass}[i]$ are the Dirac Delta position and probability mass for particle i.

[0352] The N^{th} mode/anti-mode:

[0353] The N^{th} mode of X is the particle value x at which the probability mass function f_X takes its N^{th} highest value and is calculated as $d_{pos}[i_N]$ where in is the index at which d_{mass} takes the N^{th} highest value. The N^{th} anti-mode is calculated similarly but with in being the index at which d_{mass} takes the N^{th} lowest value. If N is greater than the size of the distributional representation the statistic evaluates to a NaN ("not a number").

[0354] Distribution's support minimum or maximum value:

[0355] This calculation returns the minimum or maximum value of the Dirac delta positions of the distributional representation of X (i.e., minimum or maximum of d_{pos}).

Tail Probability:

[0356] Given a cut-off value $x_0 \in \mathbb{R}$, the calculation of the tail probability of X is

$$Pr(X > x_0) = \sum \forall\, i \in [0, K] \text{ s.t. } d_{pos}[i] > x_0 d_{mass}[i]$$

[0357] The tail probability Pr ($X \leq x0$) is calculated as: Pr ($X \leq x0$)=1.0–Pr ($X > 0$).

Loading/Storing Uncertainty Information

[0358] A load instruction in the microarchitecture loads the distributional representation that corresponds to an address of the microarchitecture's main memory to the distributional part of the destination register. A store instruction stores the distributional information of the source register to the main memory of the microarchitecture. FIG. **6**A shows how a portion of the physical memory of the processor implementing the microarchitecture stores the distributional representations. The load/store unit of the microarchitecture maps an address accessed by an application to the address that stores its distributional representation.

[0359] An additional load instruction is discussed in more detail below that initializes the distributional information of a destination register by creating a distribution representation from an in-memory array of source particle samples.

[0360] FIG. **6**C shows an implementation of the hardware module that converts source samples (e.g., sensor measurements) to the TTR representation. Nomenclature and symbology in FIG. **6**C is as follows:

dmPos[i]:	Support position of ith element of the input Dirac mixture
dmMass[i]:	Probability mass of ith element of the input Dirac mixture
dPos[i]:	Support position of the ith Dirac delta of the output TTR
dMass[I]:	Probability mass of the ith Dirac delta of the output TTR
startInd:	Lower bound of indices of the input Dirac mixture to be processed
endInd:	Upper bound of indices of the input Dirac mixture to be processed
N:	Size of output TTR representation. N is 4 for this example
←--------→	Transfer of Dirac mixture data from main memory

[0361] The module comprises multiple levels of conversion units (i.e., "Conversion unit [0,0]"; "Conversion unit [1,0]"; "Conversion unit [1,1]"; "Conversion unit [2,0]"; "Conversion unit [2,1]"; "Conversion unit [2,2]"; "Conversion unit [2,3]"), one level for each step involved in the conversion of an array of samples in the form of a SoDD (also referred to as a "Dirac mixture" herein) to a TTR representation (discussed above). A pair of integers identifies each conversion unit. The first integer corresponds to the conversion step and the second is an increasing index, e.g., "Conversion unit [1,1]" in FIG. **6**C is the second conversion unit of conversion Step 1 (Section 2.2). Because FIG. **6**C shows an example for the conversion of input samples to TTR of size four, the output level consists of four conversion units which generate the four support positions ("dmPos") and probability masses ("dmMass") of the TTR of size four. Each conversion unit has three inputs and three outputs. The first input is the memory address of the array of samples that the conversion unit will process. The microarchitecture stores the samples in the main memory as a SoDD Dirac mixture, sorted by ascending support position ("dmPos" value). The second and third inputs are two integers that correspond to the starting index ("startInd") and ending index ("endInd") of the continuous array of samples that the conversion unit will process. Each conversion unit outputs a support position ("dPos") value and a probability mass ("dMass") value. The conversion unit calculates the output probability mass as:

$$d\text{Mass}=\Sigma_{startInd}^{endInd} dm\text{Mass}[i]$$

[0362] The conversion unit calculates the mean value of the source SoDD Dirac mixture as:

$$\mu = \sum_{startInd}^{endInd} dmPos[i] * dmMass[i]$$

[0363] The conversion unit calculates the output support position as:

$$dPos = \frac{\mu}{dMass}$$

[0364] The third output of the conversion unit is an integer "kPartition" which corresponds to the index below which all sorted support positions of the input SoDD Dirac mixture are less than the calculated "dPos". In intermediate conversion levels, the conversion units propagate their output "kPartition" to the next conversion level. Depending on the subset of arrays that they must process, the "kPartition" acts as The "startInd" or "endInd" value of the conversion units of the next level. In the final conversion level, the TTR conversion module writes the output distributional information to the distributional destination register and writes the mean value of the source samples to the conventional destination register.

[0365] Given an arithmetic instruction with "Rd" target register, then the extended functional unit (**20**, **21**B) com-

putes both its particle value (according to original RISC-V ISA) and its distribution according to the distributions associated with the source registers. Every arithmetic operation applied on the extended register files (**28**A, **28**B; **29**A, **29**B), is applied equally on both on the particle (**28**A, **29**A) and distributional (**28**B, **29**B) information of the source registers. This affects the value of both the particle and distributional information of the destination register. For example, consider adding registers f1 and f2 and storing the resulting addition value in register f0:

*f*0:particle=*f*1:particle+*f*2:particle Arithmetic operation:

*f*0:distribution=*f*1:distribution+*f*2:distribution Arithmetic operation:

[0366] The results of arithmetic operations may be output to a random access memory **5** of the processor unit **4** (FIG. **6**A). These output results comprise both particle data values (f0: particle) and distributional data (f0: distribution). Each particle data value (f0: particle) is stored in a memory unit **36**, providing a physical address space, and is associated with a distribution representation (f0: distribution) stored in a distributional memory unit **37**. A memory access unit **34** and a register writeback unit **35** provide an interface between the extended register files (**28**A, **28**B; **29**A, **29**B) and the arithmetic logic units of the processor **4**. The Instruction Fetch unit **22** is in communication with the particle memory unit **36** for accessing the memory unit **36** for fetching instructions therefrom. Notably, in this example, the Load/Store unit **32** is in direct communication with the distributional memory unit **37**, but the Instruction Fetch unit **22** is not so connected. This means that the execution of arithmetic operations on distributional data may take place automatically without requiring, or interfering with, the operation of the Instruction Fetch unit **22**. In this way, the calculation of distributional data, resulting from arithmetic operations on particle data and their associated distributional data, may take place 'under the hood' at the microarchitectural level. Accordingly, the microarchitecture may be configured to allow only load/store instructions to access the random access memory **5**. In this sense, the memory unit **5** provides an extended memory, to which the microarchitecture can load and store both the particle and distributional information of the microarchitecture registers (**28**A, **28**B; **29**A, **29**B).

Tracking of Correlations Between Registers

[0367] For the improved propagation of the distributional information of the registers of the microarchitecture, it is useful to be able to track correlations between the registers. These correlations change dynamically as microarchitecture propagates the distributional information of the registers through arithmetic operations. In order to track and identify the correlation between registers, processor **4** is configured to track which memory addresses of memory unit **36** have contributed to the calculation of the value of any given floating-point or integer register at any point in time. When a particle value (f0) resulting from an arithmetic operation, is output from a register of the microarchitecture, the processor unit stores the output data of a particle value (f0) resulting from an arithmetic operation in memory when it executes a store instruction. The processor unit **4** also stores the information about the original addresses, or originating addresses, within the main memory unit **5**, of the particle data items that contributed to the output result (referred to herein as "origin addresses" or "ancestor addresses": these two terms refer to the same thing). The processor may subsequently recall the origin addresses when the contents of the register (e.g. the stored particle value (f0)) are loaded from main memory for further use. We refer to this correlation tracking as the "origin addresses tracking" mechanism.

[0368] The memory unit **5** of the processor is configured to store data items at addressed memory locations therein. The microarchitecture **4** is configured to obtain the originating memory location addresses (i.e. "origin addresses") of data items that contribute to the arithmetic operation (e.g. Arithmetic operation: f0:particle=f1:particle+f2:particle) executed by either of the floating-point or integer arithmetic logic units (floating-point ALU, **25**; Integer ALU, **27**) as the floating-point or integer arithmetic logic unit in question executes an arithmetic operation. The microarchitecture is configured to store the obtained originating memory location addresses at a storage location within the memory unit **5** and to associate that storage location with the further distribution data (e.g. f0: distribution) that is generated by the second arithmetic logic units (distributional ALU, **25**; Integer ALU, **27**).

[0369] As an illustrative example, FIG. **9** presents an example C-code **40** for calculating the Taylor series expansion of the cosine function. Line 12 of the code shows that an important part of the series expansion is the calculation of the square of C variable x. Assuming that x is a variable with distributional information, the goal of the origin addresses tracking mechanism is to detect at the microarchitectural level that variable x is multiplied with itself.

Challenges to Tracking Correlations: Ancestor/Origin Addresses Tracking Mechanism

[0370] FIG. **9** also shows the RV32IFMD ISA instructions **41** that correspond to lines 10-12 of the C-code snippet in FIG. **9**. One may observe that even in the case of the operation "x*x" the compiler requires to use two different registers (fa**4** and fa**5**) to calculate the product "x*x" (see **42** and **43**). Consequently, tracking only the IDs of the source registers involved in an instruction is not sufficient to track correlations of variables with distributional information. Despite the assignment of two different registers, the compiler in the example of FIG. **9** instructs the loading of their values from the same address. This is a core insight behind the present correlation tracking mechanism.

[0371] In the present invention, in preferred embodiments, the value of each floating-point/integer register originates from one or more address of the memory unit **5** of the processor. By maintaining and propagating these addresses the invention is able to dynamically identify correlations between any two floating-point/integer registers of the processor **4**. This information may be maintained, for example, using a dynamically linked list, which we will refer to as the "List of origin addresses" herein. An origin address can only uniquely appear in this list.

[0372] As noted above, FIG. **9** shows the C source code and RV32IFMD ISA disassembly of the calculation of the even powers of a variable x, e.g., extracted from a function that uses a Taylor series expansion to calculate the cosine of x. The compiler uses two different registers (fa**4** and fa**5**) to calculate x*x. Tracking only the identities of the source registers of instructions is not sufficient to track correlations between registers and by extension, variables with distribu-

tional information. The processor loads the values of both registers from the same memory address, which corresponds to the same variable of the source application. This is the core insight behind our autocorrelation tracking mechanism.

[0373] The value of each floating-point register originates from one or more addresses of the main memory. By keeping track of these ancestor addresses we can dynamically identify correlations between any two registers of the microarchitecture. Each time the microarchitecture executes a load instruction, it adds the source address to the set of ancestor addresses of the destination register rd. An on-chip memory of the microarchitecture is configured to store a fixed number (AAMax) of ancestor addresses for each of the architectural registers. The microarchitecture evicts addresses from this memory using a least-recently used (LRU) policy. When the microarchitecture stores a registers to memory, its set of ancestor addresses is spilled to main memory. For each arithmetic operation, if the source registers have at least one common ancestor, the microarchitecture executes the arithmetic operation in an autocorrelation-tracked manner. The microarchitecture also updates the ancestor addresses of the destination register with the union of the ancestor addresses of the source registers.

Correlation Between Register Values

[0374] Let x be a variable with distributional information. Disclosures above have introduced arithmetic operations on uncorrelated random variables. The correct calculation of expressions such as x*x is a subset of the broader problem of handling correlated (i.e., non-independent) distributions. The correct execution of such autocorrelated arithmetic operations requires point-wise operations on the support of the source operands. FIG. **6**D shows the calculation of x*x with and without the correct handling of autocorrelation. FIG. **6**D (a) shows the distribution of x created in the microarchitecture using samples from a zero-mean Gaussian distribution. FIG. **6**D (b) shows the combined outcome of uncertainty tracking with autocorrelation tracking on or off, and the ground truth of exhaustively evaluating x*x for all samples of x. The evaluation of x*x when autocorrelation tracking is on is almost identical to the ground truth. Without such handling of arithmetic on autocorrelated random variables there are negative values in the support of the outcome distribution which is incorrect for the expression x*x.

Distributional co-ALU: Example

[0375] FIG. **6**E shows the internals of the distributional co-ALU. Nomenclature and symbology in FIG. **6**E is as follows:

dmPos[i]:	Support position of the ith Dirac delta of the input register's TTR
dmMass[i]:	Probability mass of the ith Dirac delta of the input register's TTR
dPos[i]:	Support position of the ith element of the output Dirac mixture
dMass[I]:	Probability mass of ith element of the output Dirac mixture
N:	Size of output TTR representation (N is 2 in this example)
---------→	Single bit signal of the equality of two ancestor addresses
------→	Single bit signal controlling the execution of autocorrelated operations
——→	Access to the distributional register file (either integer of floating-point)

[0376] Using the information stored in the on-chip memory of active per-register ancestors (FIG. **6**A) the co-ALU determines whether the source registers have common ancestor addresses in the autocorrelation detection unit. The unit sets the output "signalsrcOperandsAutocorrelation-Signal" if it detects autocorrelation. If not set, then the co-ALU executes Algorithm 1 or Algorithm 2 (or a variant according to the instruction) shown in Tables 2 and Table 3 respectively. The smaller ALU components perform the intended arithmetic operation on the support positions of the distributional source registers. For all arithmetic operations, the co-ALU multiplies the masses of the distributional source registers. An "assertedsrcOperandsAutocorrelation-Signal" signal disables non-pointwise units and the co-ALU performs a point-to-point operation on the source positions and masses. The buffered output of both autocorrelation-tracked and uncorrelated-operand calculations is a SoDD Dirac mixture. A conversion unit like that of FIG. **6**C, converts the SoDD Dirac mixture to TTR and forwards it to the distributional register file. For the correct tracking of uncertainty in applications, The microarchitecture preferably propagates distributional information through all floating point operations. However, not all operations have the same distributional execution overhead. If one or both of the distributional source registers do not contain distributional data (i.e., are particle values) then the co-ALU performs a scaling of the support position or no operation, respectively. The quantities "src1" and "src2" may correspond to the quantities "srcVar1" and "srcVar2" noted in Algorithm 1 and Algorithm 2 shown in Tables 2 and Table 3 above, for example.

Example of Tracking Correlations

[0377] FIG. **8** is an example of the tracking of origin addresses and autocorrelation detection for the RV32IMFD assembly code snippet of FIG. **9**. The example focuses on the execution of:

$$xPower^{*} = x * x \text{ (line 12 in } FIG.\ 3)$$

[0378] Each row of the table shown in FIG. **8** corresponds to the execution of an assembly instruction of the C-code snippet of FIG. **9**. In each row, we focus on the destination register of each instruction and show its "List of origin addresses" before and after the execution of the instruction. In rows 0 and 1 of the table shown in FIG. **8**, we observe that the "List of origin addresses" of registers fa**4** and fa**5** are created to include "−72(s0)", where "s0" is a saved integer register which holds an address relative to the stack pointer. Since fa**4** and fa**5** have identical origin addresses, the processor performs an autocorrelated multiplication in row 2 of the table shown in FIG. **8**, which corresponds to the operation "x*x".

[0379] In row 3 of the table shown in FIG. **8**, a new address is loaded in fa**4**, and thus its origin address is overwritten. In row 4 of the table shown in FIG. **8**, which

corresponds to the second multiplication of the assembly C-code snippet of FIG. **9**, the origin addresses list of destination register fa**5** is set as the combination of the source registers fa**4** and fa**5**, i.e., including both addresses "−72(s0)" and "−48(s0)". In row 5 of the table shown in FIG. **8**, this origin address is stored in address location "−48(s0)" in order to be loaded at the next iteration of the for-loop of line 10 of the C-code of FIG. **9**. In this second iteration, the first instructions rows 6-8 of the table shown in FIG. **8** have the same effect as in the first iteration. In the load instruction of row 9 of the table shown in FIG. **8**, the origin address of fa**4** is updated according to the list stored in "−48(s0)". In this way, in the multiplication of row 10 of the table shown in FIG. **8**, registers fa**4** and fa**5** share a common origin ("−72(s0)") in their origin address lists and this allows the processor to identify that it needs to perform an auto-correlated multiplication. This multiplication corresponds to the operation "xPower*(x*x)". In the case of arithmetic operations with auto-correlation there is no need to store information about the distribution of the correlation of the source operands.

EXAMPLE APPLICATIONS

[0380] FIG. **10** shows the result of applying the arithmetic operation:

Arithmetic "Particle" Operation:

$$\text{particle}(6.0) = \text{particle}(5.0) + \text{particle}(1.0)$$

Arithmetic "Distribution" Operation:

$$\text{distribution}\left(N\left(6.0, \sigma_2^2\right)\right) = \text{distribution}\left(N\left(5.0, \sigma_1^2\right)\right) + \text{distribution}\left(N\left(1.0, \sigma_1^2\right)\right)$$

[0381] In which the particle values being added are 1.0 and 5.0. Each of these particles is associated with distributional information representing uncertainty in the value of the particle. For both particles, the distributional information is a Gaussian distribution (**50**, **51**) with variance σ_1^2 and with mean values of 1.0 for one particle and 5.0 for the other particle. The result is a third particle (the product) of value 6.0 and distributional information representing uncertainty in the third particle value as a Gaussian distribution **52** with variance σ_2^2 and with a mean value of 6.0. In general, the value of the variance σ_2^2 of the resulting distribution **52** will differ from the values of the variances σ_1^2 of the distributions (**50**, **51**) contributing to the sum, as can be seen by visual inspection of FIG. **10**.

[0382] In the following we present examples of the ability of the present invention, in any aspect, to correctly represent distributions from particle samples and propagate the distributional information through arithmetic operations. We compare against (i) the Monte Carlo for propagating uncertainty, and (ii) the NIST uncertainty machine [4].

Example 1: Using Samples from Parametric Distributions

[0383] We evaluate the ability of an embodiment of the present invention to represent and operate on samples from known parametric distributions. We randomly sample from Gaussian distributions to create the independent variables with distributional information "A" and "B" and perform all the basic arithmetic operations (addition, subtraction, multiplication and, division) on them. We verify the correctness of the mathematical operation by exhaustively performing each mathematical operation on all combination of samples of the input distributions (Monte Carlo). For these examples we used the TTR representation described above, with 8 Dirac deltas to represent variables with distributional information.

[0384] FIG. **11** and FIG. **12** show the histograms **61** (denoted in the figure legends as "X") of the results. In the cases that the input independent samples come from the same parametric distribution, we illustrate the histograms of "A" and "B" using the same subfigure (e.g., FIG. **11**(*a*)). We annotate the results **62** of the Monte Carlo simulation with a dashed line (MC). We show the mean **60** of each distribution in the legend of each subfigure. We also use a vertical line to annotate the mean of a distribution histogram **61** according to the invention. The x-axis of all subfigures corresponds to the domain of the distributions of each subfigure. FIGS. **11** to **12** correspond to modelling and arithmetic operations on Gaussian distributions, $(\mu;\sigma^2)$.

[0385] FIG. **11** shows the representations and arithmetic operations on two independent Gaussian random variables (N(1;1) and N(1;1)). The x-axis of all subfigures is the domain of the distribution. The dashed line shows the distribution of each arithmetic operation calculated using Monte Carlo on the input distribution samples. FIG. **12** shows the representations and arithmetic operations on two independent Gaussian random variables (N(1;1) and N(2;1).

Example 2: Thermal Expansion Coefficient

[0386] FIG. **13** shows the results for the distribution of the thermal expansion coefficient K of a cylindrical copper bar calculated from artificially-created measurements of its initial length L_a, final lengths L_b and the temperature difference ΔT. We assume that L_a is uniformly distributed on the interval [9,10], L_b is uniformly distributed on the interval [11;12] and ΔT has Gaussian distribution with N(2;1). The equation for calculating the thermal expansion coefficient is:

$$K = \frac{L_b - L_a}{L_a \Delta T}$$

[0387] The results **80** (denoted in the figure legends as "X") of the calculation of distributional information for the thermal expansion coefficient K, are compared against results (**81**, **82**) from the NIST uncertainty machine [4]. The NIST Uncertainty Machine (NISTUM) is a Web-based software application that allows users to specify random variables and perform mathematical operations on them, including complex functions. Users specify the type and parameters of the parametric distributions that the input random variables to the NISTUM follow. The NISTUM provides two methods for propagating uncertainty during arithmetic operations. One method is based on the propagation of the centralized moments of the distribution, which we will refer to as NIST uncertainty propagation expression (NIST UPE—see curve **82**). Another method is based on Monte Carlo simulation of the arithmetic operations on

distributions, which we will refer to as NIST Monte Carlo (NIST MC-see curve **81**). We examined three applications drawn from the examples of the NISTUM user manual [4]. We used random number generators to derive samples of the parametric distributions that the input variables of the examples follow. We used the Web-based interface of NISTUM to execute the example applications and get the data produced by NISTUM for both its uncertainty propagation methods. For these examples we used the TTR representation described above, with 8 Dirac deltas to represent variables with distributional information.

[0388] This example application corresponds to the measurement of the linear thermal expansion coefficient of a cylindrical copper bar of initial length L_a=1.4999 m measured with the bar at temperature T_0=288.15 K. The final length of the bar L_a was measured at temperature T_1=373.10 K and yielded L_b=1.5021 m. The measured variables were modelled as Student's t random variables with 3 degrees of freedom, with means equal to the measured values, and standard deviations equal to 0.0001 m for L_a, 0.0002 m for L_b, 0.02 K for T_0, and 0.05 K for T_1, respectively. FIG. **13** shows the comparison of the results according to the invention and NISTUM for the thermal expansion coefficient application.

Example 3: Speed Estimation Using GPS Data

[0389] In human activity tracking, it is common to use GPS coordinates to estimate a person's movement speed. For this example we use a public dataset of human activity with timestamped GPS coordinates and GPS accuracy values. We adopt the methodology of Bornholt et al. [ref. 7] to derive a distribution of the GPS coordinates around a pair of GPS longitude and latitude values of the human activity dataset. We use the distributions of GPS coordinates to calculate the distribution of the estimated speed between two GPS points (gps-speed). Compared to the conventional microarchitecture, the average improvement of the TTR in terms of Wasserstein distance from the Monte Carlo evaluation output is 1.9×.

Example 4: Bosch BME680 Sensor Conversion Routines

[0390] The BME680 by Bosch is a temperature, pressure, and humidity sensor. Bosch provides routines for converting raw ADC values to meaningful measurements using 20 sensor-specific calibration constants. This example evaluates the effect of noise in the ADC measurements and uncertainty in the calibration constants on the calibrated temperature, pressure, and humidity outputs of official commercial calibration firmware code provided by Bosch [ref. 8]. FIG. **13**B shows the noisy temperature ADC measurements and FIG. **13**C shows their effect on the converted temperature. In this application, aleatoric uncertainty leads to a bimodal distribution of the output result. The conventional architecture output falls in a zero probability range, meaning that average-filtering the noisy ADC leads to an incorrect result. Compared to the conventional methods, the present microarchitecture achieves on average 5.7× (up to 10× with TTR-64) smaller Wasserstein distance from the Monte Carlo output.

Example 5: Brown-Ham Dislocation Model

[0391] This example calculates the cutting stress of an alloy precipitate using the Brown-Ham dislocation model [Ref. 10]. Anderson et al. [Ref. 9] provide empirical value ranges for the inputs of the dislocation model. We assume that the inputs of the dislocation model follow a uniform distribution across these ranges. On average, in comparison to the conventional methods, the present microarchitecture achieves 3.83× (up to 9.3×) higher accuracy with respect to the Monte Carlo simulation.

Example 6: One-Dimensional Finite Element Model

[0392] An engineering problem for which uncertainties in the problem parameters have important safety implications relates to measurements of the properties of structural components. Consider a beam (e.g., concrete or steel) of cross-sectional area A and length L made of material with Young's modulus E loaded with an applied axial load q_0 per unit length. The beam's extension u(x) at position x from the cantilever, is:

$$u(x) = \frac{q_0}{2AE}\left(2xL - x^2\right)$$

[0393] Because of material variabilities or variations in atmospheric conditions, the Young's modulus may be uncertain and this uncertainty can be quantified with a probability distribution for the Young's modulus rather than using a single number. In the same way that engineers will often want to evaluate a floating-point implementation of the equation for the extension u(x) rather than one where the parameters are constrained to integer values, it is useful to evaluate the equation for the extension u(x) with parameters such as the Young's modulus as distributions representing uncertainty. The implementations of analytic models such as the equation for the extension u(x) or their finite-element counterparts may be in legacy or third-party libraries, making it attractive to have methods for tracking uncertainty that work on existing program binaries. A one-dimensional finite element model was used to calculate the extension of a beam when the model parameters have epistemic uncertainty. We assumed a uniform distribution of the Young's modulus input parameter of the benchmark and fixed particle values for the rest of the input parameters. The present microarchitecture achieved an average accuracy improvement of 2× compared to the conventional methods.

Example 7: Accelerated Variational Quantum Eigensolver

[0394] This quantum algorithm calculates ground states of a quantum system Hamiltonian, H [refs. 11 and 12]. The present microarchitecture was applied configured to find the quantum state $\psi(k)$ that minimizes the eigenvalue: $\langle \psi(k)|H|\psi(k)\rangle$. Typically, this uses rejection sampling to calculate $P(\phi|E)$ from measurements of $P(E|\phi)$ from a quantum phase estimation (QPE) circuit [Ref. 13]. With the present microarchitecture we can explicitly calculate the posterior from $P(E|\phi)$ and calculate $P(\phi)$. Over multiple iterations on the present microarchitecture, the algorithm can calculate better estimates of $P(\phi|E)$. FIG. **13**D shows the input which we generate using samples from a uniform distribution. FIGS. **13**E and **13**F show the decrease in the posterior variance $P(\phi)$ after two and five iterations on the present microarchitecture, respectively. The present microarchitec-

ture achieves an accuracy improvement of 6.23× compared to the conventional techniques (up to 41.3× with TTR-256).

EXAMPLE

[0395] Referring once more to FIG. **1**, in an alternative aspect of the invention, or in addition to aspects disclosed herein, an apparatus **1** is shown according to an embodiment of the invention. The apparatus comprises a computing apparatus **3** which is connected in communication with a measurement device **2** in the form of a sensor unit (e.g. an accelerometer, a magnetometer etc.) which is configured to generate measurements of a pre-determined measurable quantity: a measurand (e.g. acceleration, magnetic flux density etc.). The computing apparatus **3** includes a processor unit **4** in communication with a local buffer memory unit **5** and a local main memory unit **6**. The computing apparatus is configured to receive, as input, data from the sensor unit representing measurements of the measurand (e.g. acceleration, magnetic flux density etc.), and to store the received measurements in the local main memory unit **6**. The processor unit **4**, in conjunction with the buffer memory unit **5**, is configured to apply to the stored measurements a data processing algorithm configured for sampling the sensor measurements stored in main memory, so as to generate a sample set of measurements which represents the uncertainty in measurements made by the sensor unit **2** while also representing the measurand as accurately as the sensor unit may allow. The processor unit **4** may be configured to use the sample set to generate a probability distribution, and distributional information encoding the probability distribution, which can be used as the representation of uncertainty in measurements by the sensor unit. This representation may be according to any of the methods disclosed herein (e.g. a SoDD-based representation; a CMR representation) and discussed in detail elsewhere in the present disclosure.

[0396] The computing apparatus is configured to store the distributional data representing the uncertainty of measurements, once generated, in its main memory **6** and/or to transmit (e.g. via a serial I/O interface) the sample set to an external memory **7** arranged in communication with the computing apparatus, and/or to transmit via a transmitter unit **8** one or more signals **9** conveying the sample set to a remote receiver (not shown). The signal may be transmitted (e.g. via a serial I/O interface) wirelessly, fibre-optically or via other transmission means as would be readily apparent to the skilled person.

[0397] In this way, the computing apparatus is configured to generate and store any number (plurality) of sample sets, over a period of time, in which the distribution of measurement values within each sample set represents the probability distribution of measurements by the sensor unit **2**, and associated approximating probability distributions and encoded distributional information. Similarly, the computing apparatus is configured to generate and store any number (plurality) of sample sets, approximating probability distributions and encoded distributional information, as generated by/in different modes of operation of the sensor unit **2** or as generated by a different sensor unit **2**. In other words, a first sample set stored by the computing apparatus may be associated with a first sensor unit and a second sample set stored by the computing apparatus may be associated with a second sensor unit which is not the same as the first sensor unit. For example, the first sensor unit may be a voltage sensor (electrical voltage) and the second sensor unit may be a current sensor (electrical current). In both cases, the computing apparatus may store distributions of measurement values made by each sensor, respectively, which each represent the probability distribution of measurements by that sensor unit **2**.

[0398] The computing apparatus **3** is configured to implement a method for computation on the distributions of data stored in the main memory unit such as: sample sets, approximating probability distributions and encoded distributional information, as follows.

[0399] Referring to FIG. **14**, there is shown examples [graphs: (a), (b), (c) and (d)] of the cumulative probability distributions of sets {n} of n values within each one of four different obtained data sets. FIG. **14** also shows a sub-set **90** of values (V_i) from an obtained data set together with probability (p_i) values **91** each defining a probability for the occurrence of each value within the sub-set in question. Tuples **92** formed from data values **90** and associated probability values **91** are shown. An approximating probability distribution **93** is shown comprising a series of Dirac delta functions (SoDD representation, as disclosed herein) generated using these tuples **92**. Each value (V_i) within the sub-set (bounded to n values) is a value that the data can take on that has the highest probability within the obtained set of data.

[0400] By having the highest probability, this may mean that the data items in question comprise a pre-set number (n, an integer >1) of data items having the highest probability of occurrence within the obtained data set. In other words, the 'top' n data items that have the greatest probability or frequency, whatever that probability (frequency) may be, of occurrence within the obtained data set. This approach fixed the size of the sub-set to the value of n. The value of n may be chosen as desired according to the characteristics of the obtained data set. For example, the value of may be a value in the range: $5<n<100$, such as $n=50$, or $n=20$ etc. Other values of n may be used, of course, as appropriate. The size of the value n can be chosen in each instance of the implementation of the invention, allowing implementations that trade the hardware or software cost of implementation for representational accuracy. The approximating probability distribution **93**, comprising Dirac deltas, are an example representation where the n values represent the positions of n Dirac deltas and the probabilities associated with the values present the probabilities associated with the Dirac deltas.

[0401] Alternatively, by having the highest probability, this may mean that the data items in question comprise only those data items having probabilities, p_i, (or frequencies) exceeding a pre-set probability threshold value (e.g. $p_{threshold}=0.2$). This means that all of the probability (p_i) values 11 satisfy the condition: $p_i>p_{threshold}$. The value of $p_{threshold}$ may be chosen as desired according to the characteristics of the obtained data set. For example, the value of $p_{threshold}$ may be a value in the range: $0.1<p_{threshold}<0.9$, or in the range $0.1<p_{threshold}<0.5$, or in the range $0.2<p_{threshold}<0.5$, such as e.g. $p_{threshold}=0.25$. Other values/ranges of $p_{threshold}$ may be used, of course, as appropriate. In this case, the number (n) of values (V_i, i=1, . . . , n) in the sub-set **90** will not be fixed, in general, and will typically vary depending on the value of $p_{threshold}$. Another alternative, by having the highest probability, this may mean that the selected data items comprise the greatest number of data items have respective probabilities that are individually higher than the

probability associated with any non-selected data item (outside the set of selected data items) and the probabilities of the selected data items sum to an aggregate probability that does not exceed the threshold probability.

[0402] FIG. **14** shows real-world examples of collections of values of data items representing the distribution of values taken on by all program variables of types unsigned int and unsigned char for the programs from the MiBench suite of representative embedded system applications and from the SPEC CPU 2000 suite of representative desktop computer applications. The probabilities (Pr({n})) shown in the graphs (a), (b), (c) and (d) correspond to the probabilities of sets {n} of n values, and the fraction of the probability mass of the overall distribution that they represent, for two real-world sets of programs (MiBench [see ref. [5]], top row of plots (a) and (b); and SPEC CPU 2000 [see ref. [6]], bottom row of plots (c) and (d)). The sub-set **90** of values (V_i) shows the set of actual elements, for n=16, for the MiBench unsigned int case.

[0403] Referring to FIG. **15**, there is shown a hardware block diagram of one embodiment of the invention. The computer apparatus **3** may be configured accordingly. The apparatus is configured to obtain input data as follows.

[0404] The apparatus receives as input **100***a* a data set of numeric values (e.g., a set such as {1, 14, 2.3, 99, −8, 6}) or categorical values (e.g., a set of strings such as {"string1", "string 2" . . . }) and from them computes **101** relative frequencies of occurrence of elements within in input data set therewith to obtain a set of tuples (V_i, p_i), of values and associated probabilities of each element within the set. Alternatively, the apparatus may receive as input **100***b* a set of tuples (V_i, p_i) of values and associated probabilities that have been determined by the apparatus previously. In addition, as a further input, the apparatus may optionally receive a value of the number "n" representing the number of data elements to be contained within a sub-set {n} of data elements selected from the input data set **100***a*, or sub-set {n} of tuples from within the input **100***b* a set of tuples (V_i, p_i), by selecting **102** the tuples with the n highest probability values (p_i) within the obtained data set (n, e.g. n=16 as in FIG. **14**). In some implementations, the input parameter value **100***c* of the number "n", may be omitted in those cases where, as discussed above, the data elements are selected based on the probability threshold of each individual item, or where data elements are selected such that their aggregate probability sums to some threshold. In an alternative implementation, the apparatus may, at this stage, select a sub-set the data items within the obtained data set which have a probability of occurrence within the obtained data set which exceeds the value of a pre-set threshold probability (as discussed above).

[0405] Next, the apparatus normalises the probability values (p_i) of each respective tuple (V_i, p_i) associated with selected data items, such that the sum of those probabilities, of all selected data items, is equal to 1.0. This normalisation re-scales the probability values to be relative probabilities of occurrence amongst the respective tuples (V_i, p_i) for members of the sub-set of data items. Each tuple (V_i, p_i) for each of the data items within the sub-set comprises a first value (V_i) and a second value (p_i) wherein the first value is a value of the respective data item and the second value is a value of the normalised probability of occurrence of the respective data item within the obtained data set.

[0406] The contents of each one of the 'n' tuples (V_i, p_i) are then stored, separately, in a memory table **110** and in a distribution memory **108**. In particular, for each tuple (V_i, p_i), the first value (V_i) thereof is stored at a respective memory location (L_i) of the distribution memory **108**. However, the second value (p_i) of the same tuple is stored in a sub-table (**103**: in software or hardware) of the memory table **110** in association with a pointer **115** configured to identify the respective memory location (L_i) of the first value (V_i) of the tuple (V_i, p_i). This means that the sub-table **103** contains only the probability value **106** of each tuple, and a pointer **115** to the location **107** associated data value (numeric or categorical) of the tuple within a separate memory **108**. The memory table **110** may be a structured memory in a circuit structure rather than an array in a computer memory. The apparatus includes a logic unit (**104**, **105**) configured for receiving the selected 'n' tuples **102** and to store the probability values (p_i), in association with respective pointers **107**, at locations within a sub-table of the table if the data values (V_i) of the tuples comply with criteria defined by the logic unit. In the way, the apparatus may structure the memory table **110** to comprise sub-tables **103** each of which contains data of tuples satisfying a particular set of criteria. The criteria defining any one of the sub-tables may, of course, be different to the criteria defining any of the other sub-tables.

[0407] Computation of arithmetic operations may be executed on these distributions, in any manner described herein, by a microprocessor, digital signal processor (DSP) or Field Programmable Gate Array (FPGA) of the apparatus **31**.

[0408] The apparatus is configured to implement a copula structure **109** for combining the individual distributions **93** represented using the tuples of separate tables (each formed as described above, e.g. SoDD), or of separate sub-tables, to achieve joint distributions. The copula structure may be configured to generate a multivariate cumulative distribution function for which the marginal probability distribution of each variable is uniform on the interval [0, 1]. This is an example only, and it is to be understood that using the copula to generate a multivariate CDF from uniform marginals is optional. There are many parametric copula families available to the skilled person for this purpose.

[0409] The logic unit may be configured to implement a probabilistic (e.g. non-Boolean) predicate **105** for the table which is configured to return a probability (Bernoulli(p)) (i.e. rather than a Boolean 'true/'false' value), for each of the given data values (V_i) to which it is applied. A schematic example is shown in FIG. **16**. The criteria defining different sub-tables may be implemented in this way. Since the predicate may be considered in terms of a predicate tree, this permits the predicate to be flattened into a string using pre-order traversal or post-order traversal of all of the nodes of the tree and this flattened tree can be used as the distribution representation, if desired. This may significantly reduce the size of the representation to being one that scales linearly with the size of the representation rather than exponentially. Techniques for pre-order traversal or post-order traversal of all of the nodes of the tree may be according to techniques readily available to the skilled person.

[0410] The apparatus may be further configured to determining a probability (or frequency) of occurrence of the data items within the obtained data set which do not exceed the

value of the pre-set threshold probability (or threshold frequency) or are not amongst the pre-set number {n}. In other words, the method may include determining the further probability ($p_{overflow}$) of occurrence of data items within the obtained data set that do not belong to the sub-set {n} of data items or of selected tuples with the n highest probability values. In this case, the apparatus normalises the probability values (p_i), of each of the {n} selected data items such that the sum:

$$p_{overflow} + \sum_{i=1}^{n} p_i = 1.0.$$

[0411] In this way, the further probability may take account of the probability of a data item of the obtained data set being outside of the sub-set and thereby provide an 'overflow' probability.

[0412] The apparatus, in this case, generates a collective tuple ($V_{overflow}$, $p_{overflow}$) collectively representing those data items of the obtained data set not within the sub-set. The value $V_{overflow}$, is collectively representative of the data items of the obtained data set not within the sub-set. The value $p_{overflow}$, is a value of the normalised further probability of occurrence (e.g. cumulative or aggregate probability) of the data items not within the selected sub-set, {n}. The collective tuple may be stored in the memory table **110**, split across the sub-table **103** and the distribution memory **108**, in the manner described above. In a preferred embodiments, $V_{overflow}$ may be a "special" value, analogous to NaN ("not a number") and Inf (infinity) which denotes an unknown value. Such values are sometimes referred to as erasure values.

[0413] An insight of the present invention is to represent the elements of a sub-set {n} with a hardware table structure **110** comprising a table entries for each of: (1) pointers to memory locations for the basis values of the distributions, whether numbers, strings, etc., including a possible overflow item to represent all other possible basis values not explicitly listed; (2) table entries for the probabilities for each basis value, including an entry for the probability of the overflow item mentioned above; (3) updating the columns of the table to ensure the probabilities sum to one. Because each distribution in use in the computing system could in principle be associated with a different such table (e.g., distributions corresponding to different points in time of a heteroscedastic process or distributions corresponding to different variables in a program), from the context of the overall computing system, the present disclosure refers to these tables as sub-tables. In a typical implementation, a system may contain a collection of such sub-tables, one sub-table for each distribution that the system requires to represent.

[0414] Every distribution instance (i.e., each sub-table) may be given a unique identifier, so that a computing system can reference the different distributions (i.e., the marginals).

[0415] Because the structure disclosed herein uses pointers to elements as the value representation, rather than the values themselves, an implementation can represent distributions where the basis values might either represent specific integers, specific real-valued values, or integer or real-valued value ranges. They could also represent strings or other categorical labels. In the special case where the values to be represented can fit directly into the table, the pointers can be replaced with the values to be represented, directly. The ability of the invention to use entries in the table to represent value ranges means that representing histograms of numbers can be used. The invention also provides the flexibility to represent, e.g., individual strings of text or lexicographically-ordered ranges of strings of text in a language orthography and their associated probabilities, and so on.

[0416] In addition to operations on the distribution representation in a microprocessor that permits arithmetic on distributions on numbers, the operations on the sub-table could also be set theoretic operations such as intersection, union, complement, and so on. For example, given two distributions representing sets of words and their corresponding probabilities, the union operation will have a new distribution whose basis values are the n items (for a representation table of size n) with the highest probabilities from the constituent sets of the union operation, with the probabilities appropriately normalized so that the new distribution has elements whose probabilities sum to one (1.0).

[0417] The components of the hardware structure maybe sub-tables for each distribution, a memory for holding the basis values of distributions, a copula structure for combining the individual distributions to achieve a joint distribution, and logic for taking sets of values and placing them into the sub-tables in the form of distributions.

EXAMPLE

[0418] In computer science, three-address code (often abbreviated to TAD, TAC or 3AC) is an intermediate code used by optimizing compilers to aid in the implementation of code-improving transformations. Each TAC instruction has at most three operands and is typically a combination of assignment and a binary operator. For example, t1:=t2+t3. The name derives from the use of three operands in these statements even though instructions with fewer operands may occur. In three-address code, a program may be broken down into several separate instructions. These instructions translate more easily to assembly language.

[0419] A basic block in a three-address code (TAD) can be considered to be a sequence of contiguous instructions that contains no jumps to other parts of the code. Dividing a code into basic blocks makes analysis of control flow much easier. In compiler construction, a basic block may be considered as a straight-line code sequence with no branches in, except to the entry to the code, and no branches out, except at the exit from the code. This makes a basic blocks highly amenable to analysis. Compilers usually decompose programs into their basic blocks as a first step in the analysis process. Basic blocks can also be considered to form the vertices or nodes in a control flow graph.

[0420] FIG. **22** shows an example of TAD of the following program for generating a 10×10 diagonal matrix with diagonal matrix elements Xi.

Example: Code to Set a 10×10 Diagonal Matrix with Diagonal Elements Xi

```
for i from 1 to 10 do
  for j from 1 to 10 do
    a[i, j] = 0.0;
  for i from 1 to 10 do
    a[i, i] = Xi;
```

[0421] The TAC is partitioned into basic blocks B1 to B6 according to the partitioning rules defined below. Here, Xi appears in basic block B6, and the value of Xi may be generated by another basic block of code (Block B7: see FIG. **19**, FIG. **20** and FIG. **21**) designed to calculate the following:

$$Xi = f(i, x1, x2, x3) = \left(\frac{[x1 + x2]}{[x3]^2[x1 - x2]}\right)$$

[0422] Steps (3)-(6) of the TAD are used to make a matrix element '0' and step (15) of the TAD is used to make a matrix element Xi.

Partitioning Three-Address Code into Basic Blocks

[0423] The process of partitioning a TAD code comprises an input stage of receiving a TAD code, followed by a processing stage in which the input TAD code is processed to partition it, as follows.

Input:

[0424] A sequence of three address instructions (TAD).

Process:

[0425] "Leader" instructions within a code are determined as follows:

[0426] (1) The first "three-address" instruction of the code is a leader.

[0427] (2) Instructions which are targets of unconditional or conditional jump/goto statements are leaders.

[0428] (3) Instructions which immediately follows unconditional or conditional jump/goto statements are leaders.

[0429] For each leader, its basic block contains itself and all instructions up to, but excluding, the next leader. A basic block contains only statements that execute in a sequence one after the other. A variable name in a basic block is 'live' at a given point if its value is used after that point in the program.

[0430] Application of this process to the TAD code of FIG. **22** results in a first basic block B1 comprising a first leader ("Leader 1") in the form of the first statement line 1) of the TAD. The second basic block B2 arises at the second statement line 2) of the TAD which defines a second leader ("Leader 2"). This arises because this statement line is the target of a "goto" statement at TAD line 11). The third basic block B3 arises at the third statement line 3) of the TAD which defines a third leader ("Leader 3"). This arises because this statement line is the target of a "goto" statement at TAD line 9). The fourth basic block B4 arises at the tenth statement line 10) of the TAD which defines a fourth leader ("Leader 4"). This arises because this statement line immediately follows a conditional "goto" statement at TAD line 9). The fifth basic block B5 arises at the twelfth statement line 12) of the TAD which defines a fifth leader ("Leader 5"). This arises because this statement line immediately follows a conditional "goto" statement at TAD line 11). The final basic block B6 of the TAD code arises at the thirteenth statement line 13) of the TAD which defines a sixth leader ("Leader 6"). This arises because this statement line is the target of a "goto" statement at TAD line 17). In this way, a TAD code may be partitioned into basic blocks.

[0431] The following disclosure describes an example of a computer apparatus configured for generating a TAD code in respect of program codes executed by the computer, and configured for partitioning the TAD codes into basic blocks. The computer apparatus is configured rearrange TAD instruction sequences to reduce numeric error in propagating uncertainty across the computation state (e.g., registers) on which the instruction sequences operate. This is particularly useful when executing arithmetic on distributions representing uncertainty in data handled by the computer, as discussed more fully elsewhere herein. The data handled by the computer may, for example, be measurement data from a measurement apparatus (e.g. a sensor) and the distributions representing uncertainty in data may represent the uncertainty in the value of the measurements made by the measurement apparatus.

[0432] Referring once more to FIG. **1**, in an alternative aspect of the invention, or in addition to aspects disclosed herein, an apparatus **1** is shown according to an embodiment of the invention. The apparatus comprises a computing apparatus **3** which is connected in communication with a measurement device **2** in the form of a sensor unit (e.g. an accelerometer, a magnetometer etc.) which is configured to generate measurements of a pre-determined measurable quantity: a measurand (e.g. acceleration, magnetic flux density etc.). The computing apparatus **3** includes a processor unit **4** in communication with a local buffer memory unit **5** and a local main memory unit **6**. The computing apparatus is configured to receive, as input, data from the sensor unit representing measurements of the measurand (e.g. acceleration, magnetic flux density etc.), and to store the received measurements in the local main memory unit **6**. The processor unit **4**, in conjunction with the buffer memory unit **5**, is configured to apply to the stored measurements a data processing algorithm configured for sampling the sensor measurements stored in main memory, so as to generate a sample set of measurements which represents the uncertainty in measurements made by the sensor unit **2** while also representing the measurand as accurately as the sensor unit may allow. The processor unit **4** is configured to use the sample set to generate an approximating probability distribution and distributional information encoding the approximating probability distribution as the representation of uncertainty in measurements by the sensor unit according to any of the methods disclosed herein (e.g. a SoDD-based representation; a CMR representation as disclosed in other aspects herein) and discussed in detail elsewhere in the present disclosure.

[0433] The computing apparatus is configured to store the distributional data representing the uncertainty of measurements, once generated, in its main memory **6** and/or to transmit (e.g. via a serial I/O interface) the sample set to an external memory **7** arranged in communication with the computing apparatus, and/or to transmit via a transmitter unit **8** one or more signals **9** conveying the sample set to a remote receiver (not shown). The signal may be transmitted (e.g. via a serial I/O interface) wirelessly, fibre-optically or via other transmission means as would be readily apparent to the skilled person.

[0434] In this way, the computing apparatus is configured to generate and store any number (plurality) of sample sets, over a period of time, in which the distribution of measurement values within each sample set represents the probability distribution of measurements by the sensor unit **2**, and

associated approximating probability distributions and encoded distributional information. Similarly, the computing apparatus is configured to generate and store any number (plurality) of sample sets, approximating probability distributions and encoded distributional information, as generated by/in different modes of operation of the sensor unit **2** or as generated by a different sensor unit **2**. In other words, a first sample set stored by the computing apparatus may be associated with a first sensor unit and a second sample set stored by the computing apparatus may be associated with a second sensor unit which is not the same as the first sensor unit. For example, the first sensor unit may be a voltage sensor (electrical voltage) and the second sensor unit may be a current sensor (electrical current). In both cases, the computing apparatus may store distributions of measurement values made by each sensor, respectively, which each represent the probability distribution of measurements by that sensor unit **2**.

[0435] The computing apparatus **3** is configured to implement a method for computation on the distributions of data stored in the main memory unit such as: sample sets, approximating probability distributions and encoded distributional information, as follows.

[0436] Referring to FIG. **17**, there is shown a flow diagram illustrating process steps in the method implemented by the computer apparatus **3**. The process steps are configured for computing a numerical value for uncertainty in the result of a multi-step numerical calculation comprising a sequence of separate calculation instructions defined within a common "basic block".

[0437] The method is applied to distributions of data contained within the buffer memory unit **5** of the computer apparatus **3**, having been placed in the buffer memory from the main memory unit **6** by the processor unit **4**, for this purpose.

[0438] Initialisation: The initial state of the system consists of uncertainty representations for all live registers (registers whose values will be read again before they are overwritten).

[0439] Step #1: Receive instruction sequence seq and place it in buffer memory unit **5**.

[0440] Step #2: Split seq into basic blocks, sequences of instructions with no intervening control-flow instruction.

[0441] Step #2B: For each basic block, form one expression for the uncertainty of each variable that is written to but not overwritten before the exit of the basic block.

[0442] Step #2C: Simplify each expression obtained in the previous step into the smallest number of terms possible and then propagate the uncertainty through this simplified expression.

[0443] Step #3: Rearrange instructions in each basic block based on their dependencies if necessary, and at the end of each basic block update the uncertainty of each register that is written to but not overwritten before the exit of the basic block.

[0444] Step #4: Output the re-ordered sequence $seq_{re\text{-}ordered}$ and updated uncertainties (distributions of data) for the sequence.

[0445] By this method (i.e. Step #1 to Step #4), the computer apparatus **3** identifies "live-out" variable(s) of the "basic block" (Step #1). It then identifies calculation instructions on who's output the value of the "live-out" variable depends (Step #2). Then, at steps #2A and #2B, it provides a mathematical expression combining the calculation instructions identified at step #2. Using the mathematical expression of provided at step #2C, the computer apparatus computes, at step #3, a numerical value for uncertainty in the "live-out" variable. Notably, the uncertainty value computed at step #3 is the uncertainty in the result of the multi-step numerical calculation. This process not only makes the final result of the calculation more accurate, but also makes the calculation process more efficient.

[0446] In this way, for the live-out variables of the basic block, rather than computing the updated uncertainty for each TAD instruction, the sequence of instructions whose result determines the value of the live-out variable can be combined into a single expression, at step #2C, for the purposes of computing the updated uncertainty of the live-out variable.

[0447] The method may be implemented in a programmable processor. It provides a means for rearranging instructions that perform operations on representations of uncertainty, such that the rearranged instructions still obey true data dependences. The rearranged instruction sequences have better numerical stability when the processor processing the instructions associates uncertainty representations with the values on which the instructions operate.

[0448] In one realization, when the uncertainty representation is based on the moments of a probability distribution (e.g. the CMR uncertainty representation disclosed herein) and where the method for computing on uncertainty is based on the Taylor series expansion of the instruction operation around the mean value of the value being operated on, then the rearrangement of instructions improves the numerical stability of the evaluation of the Taylor series expansion of the function corresponding to the aggregate set of instructions.

[0449] The rearrangement can be applied to any method for propagating uncertainty through a sequence of arithmetic operations and is not limited to this specific example case of using Taylor series expansions of the functions through which uncertainty is being propagated. The instruction rearrangement method is valuable for any programmable processor architecture that implements arithmetic on probability distributions and where that arithmetic could have different numerical stability properties when instructions are rearranged in dependence-honouring functionally-equivalent orderings.

Combining Variance Estimate for Live-Out Variables

[0450] FIG. **18**(*a*) shows a simple program that performs a sequence of arithmetic operations and FIG. **18**(*b*) shows the assembly language instruction sequence generated by a compiler, from that program. While different compilers and different compilation options will lead to slightly different instruction sequences, the sequence of program statements in FIG. **18**(*a*) will always be translated into a sequence of assembly language instructions which, when executed, lead to the arithmetic operations required by the program. In particular, FIG. **18**(*a*) shows a program that performs arithmetic on three floating-point variables. When executed on a processor architecture that associates some representation of uncertainty to each variable, the arithmetic in each program statement will require the hardware implemented in the processor to evaluate new distributions for the uncertainties of intermediate and final results. FIG. **18**(*b*) shows the

program in FIG. **18**(*a*) compiles to the sequence of assembly language instructions for a processor. Rather than updating the uncertainty for the destination register of each instruction in FIG. **18**(*b*) separately, it is possible to generate an instance of an equation defining the uncertainty (σ_y^2) in a calculated quantity, $y=f(x_1, \ldots, x_n)$, that implements the combined expression:

$$f = (x1 + x2)/(x3 * x3 * (x1 - x2)).$$

[0451] In particular, for example, let f be a function implemented by some sequence of statements in a high-level language such as C or implemented by any sequence of instructions executing within a programmable processor. The function f might also represent a collection of logic gates in a fixed-function digital integrated circuit such as an ASIC or in a field-programmable digital circuit such as an FPGA. Let $x_1, x_2, \ldots, x_n$ be the parameters of the function f.

[0452] The method for determining the uncertainty of the function f based on the Taylor series expansion of f specifies that the uncertainty in f, represented by its standard deviation as follows. This example is particularly relevant for propagating uncertainty is the CMR method disclosed herein. The present invention, in the presently-described aspect, could be applied to other distribution representations and propagation methods other than the CMR/Taylor series expansion methods. However, for illustrative purposes, and to allow a better understanding of the invention, the following example concerns the CMR/Taylor series expansion methods.

$$\text{Let } y = f(x_1, \ldots, x_n), \text{ then:}$$

$$\sigma_y^2 \approx \left(\frac{\partial f(\overline{x_1}, \ldots, \overline{x_n})}{\partial x_1}\right)^2 \sigma_{x_1}^2 + \left(\frac{\partial f(\overline{x_1}, \ldots, \overline{x_n})}{\partial x_2}\right)^2 \sigma_{x_2}^2 + \ldots + 2\sigma_{x_1 x_2}^2 \left(\frac{\partial f(\overline{x_1}, \ldots, \overline{x_n})}{\partial x_1}\right)\left(\frac{\partial f(\overline{x_1}, \ldots, \overline{x_n})}{\partial x_2}\right) + \ldots$$

[0453] The variances in the right hand side of the above expression are typically small and computing the squares and higher powers of those variances leads to numeral errors in practice. The insights of the method in this disclosure are:

[0454] (1) to represent the standard deviations within the computation hardware's internal representation for the digital logic circuit implementing the above equation (σ_y^2), with an unsigned fixed-point representation rather than representing them with floating-point representations, since the variances will always be positive; and,

[0455] (2) to combine sequences of arithmetic operations that do not contain a control-flow instruction (i.e., a basic block) where those operations being combined would otherwise be treated separately, for the purposes of computing the variance of variables that will be used again before being overwritten (i.e., the live variables).

[0456] The second of the insights above reduces the number of times the approximations of the above equation (σ_y^2) need to be applied, to reduce error in the computed uncertainty. Both of these insights can be implemented in hardware or could also be implemented as a compile-time transformation on a source program, or using a combination of compile time transformations (e.g., to determine the combined function for each basic block and its partial derivatives, e.g., using automatic differentiation) combined with evaluating the instance of the above equation (σ_y^2) generated in that process, in hardware when the values of the variances are available. Other methods for representing and propagating uncertainty in a digital computation hardware structure might use methods different from the above equation (σ_y^2) but the present methods herein make the uncertainty tracking more numerically stable and will still apply.

[0457] FIG. **19** schematically shows the effect of the invention on implementation of the program code of FIG. **18**(*a*) for calculating the quantity:

$$f = (x1 + x2)/(x3 * x3 * (x1 - x2))$$

via a TAD code in a basic block B7 both before ("BEFORE" in FIG. **19**) and after ("AFTER" in FIG. **19**) application of the invention. In the case of before application of the invention, it can be seen that the quantity σ_y^2 must be calculated eight times, once for each of the eight code lines of the TAD code within basic block B7. The final output line of the TAD code produces the variable "t8" which carries the value of f. The final output line of the TAD code produces the latest value of the quantity σ_y^2 quantifying the uncertainty in f. All preceding values of σ_y^2 quantify the uncertainty in the preceding seven TAD variables (t1 to t7) within the basic block B7. FIG. **20** shows this explicitly, in which each expression σ_y^2 for each one of the TAD code lines t1 to t8 (see "BEFORE" in FIG. **19**) are explicitly defined and calculated. Contrast this with FIG. **21** in which the invention has been applied and only the expression σ_y^2 for the final TAD code line t8 (see "AFTER" in FIG. **19**) is explicitly defined and this expression is used/calculated by the processor **4** to quantify the uncertainty in the function f. This instruction sequence has better numerical stability for calculating the uncertainty representation σ_y^2.

[0458] In other words, FIG. **21** implements Step #2B, above, whereby for basic block B7, one expression is formed for the uncertainty σ_y^2 of the variable "t8" that is written to but not overwritten before the exit of that basic block. Step #2C is implemented in that the expression σ_y^2 is a simplified expression obtained in Step #2B having the smallest number of terms possible. This is then used to propagate the uncertainty.

[0459] References herein to a "tuple" may be considered to include a reference to a finite ordered list of elements. References herein to an "n-tuple" may be considered to include a reference to a sequence of n elements, where n is a non-negative integer.

[0460] References herein to a "parameter" may be considered to include a reference to a numerical or other measurable factor forming one of a set (of one or more) that defines a system or sets its properties, or a quantity (such as a mean or variance) that describes a statistical population, or a characteristic that can help in defining or classifying a particular system, or an element of a system that identifies the system.

[0461] References herein to "threshold" may be considered to include a reference to a value, magnitude or quantity

that must be equalled or exceeded for a certain reaction, phenomenon, result, or condition to occur or be manifested.

[0462] References herein to "distribution" in the context of a statistical data set (or a population) may be considered to include a reference to a listing or function showing all the possible values (or intervals) of the data and how often they occur. A distribution in statistics may be thought of as a function (empirical or analytical) that shows the possible values for a variable and how often they occur. In probability theory and statistics, a probability distribution may be thought of as the function (empirical or analytical) that gives the probabilities of occurrence of different possible outcomes for measurement of a variable.

[0463] The features disclosed in the foregoing description, or in the following claims, or in the accompanying drawings, expressed in their specific forms or in terms of a means for performing the disclosed function, or a method or process for obtaining the disclosed results, as appropriate, may, separately, or in any combination of such features, be utilised for realising the invention in diverse forms thereof.

[0464] While the invention has been described in conjunction with the exemplary embodiments described above, many equivalent modifications and variations will be apparent to those skilled in the art when given this disclosure. Accordingly, the exemplary embodiments of the invention set forth above are considered to be illustrative and not limiting. Various changes to the described embodiments may be made without departing from the spirit and scope of the invention.

[0465] For the avoidance of any doubt, any theoretical explanations provided herein are provided for the purposes of improving the understanding of a reader. The inventors do not wish to be bound by any of these theoretical explanations.

[0466] Any section headings used herein are for organizational purposes only and are not to be construed as limiting the subject matter described.

[0467] Throughout this specification, including the claims which follow, unless the context requires otherwise, the word "comprise" and "include", and variations such as "comprises", "comprising", and "including" will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

[0468] It must be noted that, as used in the specification and the appended claims, the singular forms "a," "an," and "the" include plural referents unless the context clearly dictates otherwise. Ranges may be expressed herein as from "about" one particular value, and/or to "about" another particular value. When such a range is expressed, another embodiment includes from the one particular value and/or to the other particular value. Similarly, when values are expressed as approximations, by the use of the antecedent "about," it will be understood that the particular value forms another embodiment. The term "about" in relation to a numerical value is optional and means for example +/−10%.

REFERENCES

[0469] [1] Anders Hald, *The early history of the cumulants and the Gram-Charlier series*, International Statistical Review 68 (2000), no. 2, 137-153.

[0470] [2] Peter Hall, *The bootstrap and Edgeworth expansion*, Springer Science & Business Media, 2013.

[0471] [3] Rabi N Bhattacharya, Jayanta K Ghosh, et al., *On the validity of the formal Edgeworth expansion*, Ann. Statist. 6 (1978), no. 2, 434-451.

[0472] [4] T. Lafarge and A. Possolo, *NIST uncertainty machine—user's manual*, National Institute of Standards and Technology, Gaithersburg, 2015. (https://uncertainty.nist.gov/ and https://www.nist.gov/publications/uncertainty-machine-users-manual)

[0473] [5] MiBench: "*A free, commercially representative embedded benchmark suite*" by Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, December 2001. (http://vhosts.eecs.umich.edu/mibench//)

[0474] [6] The most widely used benchmarks are the Standard Performance Evaluation Corporation (SPEC) CPU benchmarks: "*SPEC*2000 *Retires SPEC*92", by B. Case, The Microprocessor Report, vol. 9, 1995

[0475] [7] James Bornholt, Todd Mytkowicz, and Kathryn S Mckinley. 2015. Uncertain<t>: Abstractions for uncertain hardware and software. IEEE Micro 35, 3 (2015), 132-143.

[0476] [8] Bosch Sensortec. [n. d.]. BME680 sensor API. [Online]. Available: https://github.com/BoschSensortec/BME680_driver, Accessed: Jul. 9, 2021.

[0477] [9] M. J. Anderson, F. Schulz, Y. Lu, H. S. Kitaguchi, P. Bowen, C. Argyrakis, and H. C. Basoalto. 2020. On the modelling of precipitation kinetics in a turbine disc nickel-based superalloy. Acta Materialia 191 (2020), 81-100. https://doi.org/10.1016/j.actamat.2020.03.058.

[0478] [10] L M Brown and R K Ham. 1971. Dislocation-particle interactions. Strengthening methods in crystals (1971), 9-135.

[0479] [11] James R Cruise, Neil I Gillespie, and Brendan Reid. 2020. Practical Quantum Computing: The value of local computation. arXiv preprint arXiv: 2009.08513 (2020).

[0480] [12] Daochen Wang, Oscar Higgott, and Stephen Brierley. 2019. Accelerated Variational Quantum Eigensolver. Phys. Rev. Lett. 122 (April 2019), 140504. Issue 14. https://doi.org/10.1103/PhysRevLett. 122. 140504.

[0481] [13] Nathan Wiebe and Chris Granade. 2016. Efficient Bayesian Phase Estimation. Phys. Rev. Lett. 117 (June 2016), 010503. Issue 1. https://doi.org/10.1103/PhysRevLett. 117.010503.

1-25. (canceled)

26. A computer-implemented method for the encoding of, and computation on, distributions of data, the method comprising:

obtaining a first set of data items;

obtaining a second set of data items;

generating a first tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the first set;

generating a second tuple containing parameters encoding a probability distribution characterising distribution of the data items of the second set in which the parameters used to encode the distribution of the data items of the second set are substantially the same as parameters used to encode distribution of the data items of the first set;

generating a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters

encoding a probability distribution representing the result of applying an arithmetic operation on the first probability distribution and the second probability distribution; and

outputting the third tuple.

27. The computer-implemented method of claim **26** wherein the first set of data comprises samples of a first random variable, and the second set of data comprises samples of a second random variable.

28. The computer-implemented method of claim **26** wherein the outputting of the third tuple comprises one of more of: storing the third tuple in a memory; and transmitting a signal conveying the third tuple.

29. The computer-implemented method of claim **28** comprising storing the third tuple in a local memory unit or a remote memory unit.

30. The computer-implemented method of claim **26** comprising outputting the first tuple and the second tuple by one of more of:

storing the first tuple and the second tuple in a memory;

transmitting a signal conveying the first tuple and the second tuple;

obtaining the output first tuple by one of more of retrieving the output first tuple from a memory and receiving a signal conveying the output first tuple;

obtaining of the output second tuple by one of more of retrieving the output second tuple from a memory, receiving a signal conveying the output second tuple, and, generating the third tuple using parameters contained within the obtained first tuple and within the obtained second tuple.

31. The computer-implemented method of claim **30** wherein the first tuple and the second tuple are obtained as output from a remote memory unit or received as output from a receiver in receipt of a signal conveying the first and second tuples from a remote transmitter or source.

32. The computer-implemented method of claim **26** wherein the arithmetic operation comprises one of more of: addition; subtraction; multiplication; and division.

33. The computer-implemented method of claim **26** wherein the third tuple contains parameters encoding a probability distribution characterising distribution of data items of a third set of data items in which parameters used to encode the distribution of the data items of the third set are substantially the same as the parameters used to encode the distribution of the data items of the first set.

34. The computer-implemented method of claim **26** wherein:

the first tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the first set;

the second tuple contains parameters encoding the position of data items within the probability distribution characterising the distribution of the data items of the second set; and

the third tuple contains parameters encoding the position of data items within a probability distribution characterising the distribution of the data items of a third set of data items.

35. The computer-implemented method of claim **26** wherein:

the first tuple contains parameters encoding position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the first set;

the second tuple contains parameters encoding the position and/or width of data intervals within the probability distribution characterising the distribution of the data items of the second set; and

the third tuple contains parameters encoding the position and/or width of data intervals within a probability distribution characterising the distribution of the data items of a third set of data items.

36. The computer-implemented method of claim **26** wherein:

the first tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the first set;

the second tuple contains parameters encoding the probability of data items within the probability distribution characterising the distribution of the data items of the second set; and

the third tuple contains parameters encoding the probability of data items within a probability distribution characterising the distribution of the data items of a third set of data items.

37. The computer-implemented method of claim **26** wherein:

the first tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the first set;

the second tuple contains parameters encoding the value of one or more statistical moments of the probability distribution characterising the distribution of the data items of the second set; and

the third tuple contains parameters encoding the value of one or more statistical moments of a probability distribution characterising the distribution of the data items of a third set of data items.

38. The computer-implemented method of claim **26** wherein:

the probability distribution characterising the distribution of the data items of the first set comprises a distribution of Dirac delta functions;

the probability distribution characterising the distribution of the data items of the second set comprises a distribution of Dirac delta functions; and

the probability distribution characterising the distribution of the data items of the third set comprises a distribution of Dirac delta functions.

39. The computer-implemented method of claim **26** in which the first tuple is an N-tuple, the second tuple is an N-tuple, and the third tuple is an M-tuple for which $N^2/2<M<2N^2$ in which $N>1$ is an integer.

40. A computer program product comprising a computer program which, when executed on a computer, implements the method according to claim **26**.

41. An apparatus for implementing the encoding of, and computation on, distributions of data, the apparatus comprising:

a memory for storing a first set of data items and a second set of data items; and

a processor configured to perform the following processing steps,

generate a first tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the first set;

generate a second tuple containing parameters encoding a probability distribution characterising the distribution of the data items of the second set in which the parameters used to encode the distribution of the data items of the second set are substantially the same as the parameters used to encode the distribution of the data items of the first set;

generate a third tuple using parameters contained within the first tuple and using parameters contained within the second tuple, the third tuple containing parameters encoding a probability distribution representing the result of applying an arithmetic operation on the first probability distribution and the second probability distribution; and

output the third tuple.

42. An apparatus according to claim **41** wherein the processor is a microprocessor.

43. An apparatus according to claim **41** wherein a microarchitecture is implemented in software of within the hardware of a microprocessor, FPGA, or other digital computation device, for representing probability distributions of items including both numeric values and categorical values.

44. An apparatus according to claim **41** wherein a microarchitecture provides a non-intrusive architectural extension to the RISC-V ISA, for computing with both epistemic and aleatoric uncertainty.

45. An apparatus according to claim **41** wherein a microarchitecture executes existing RISC-V programs unmodified and whose ISA is extended to expose new facilities for setting and reading distribution data without changing program semantics.

* * * * *