F-Test of Equality of Variances - TECH

AUGUSTO MARTINS – 01656520

2018S - ADVANCED TOPICS IN PARALLEL COMPUTING

UNIVERSITY OF VIENNA

Implementation Steps

1

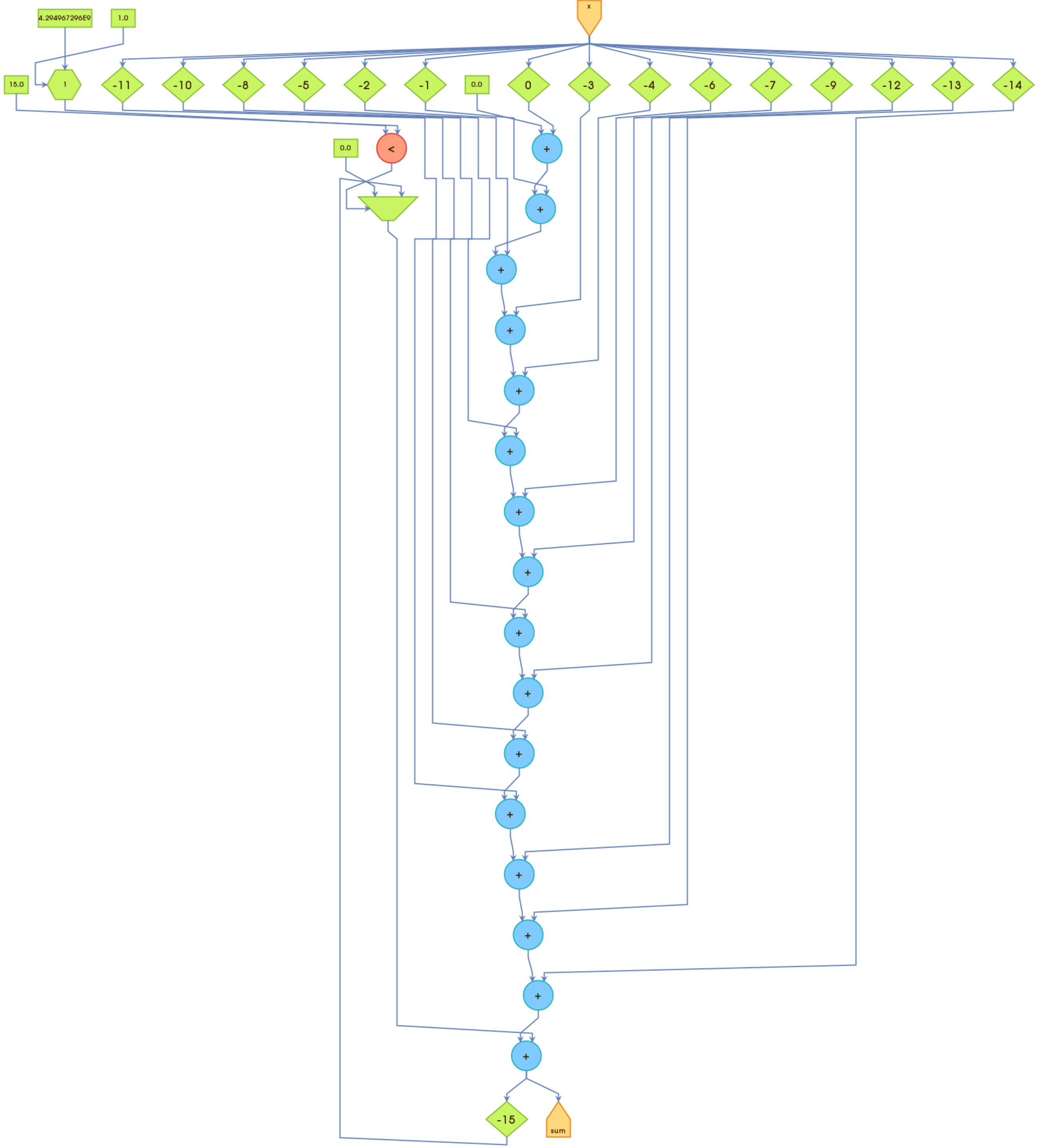
How to efficiently implement an accumulator?

2

Rewrite variance expression to better fit the dataflow paradigm.

3

Defining the F-test.



Accumulator Kernel

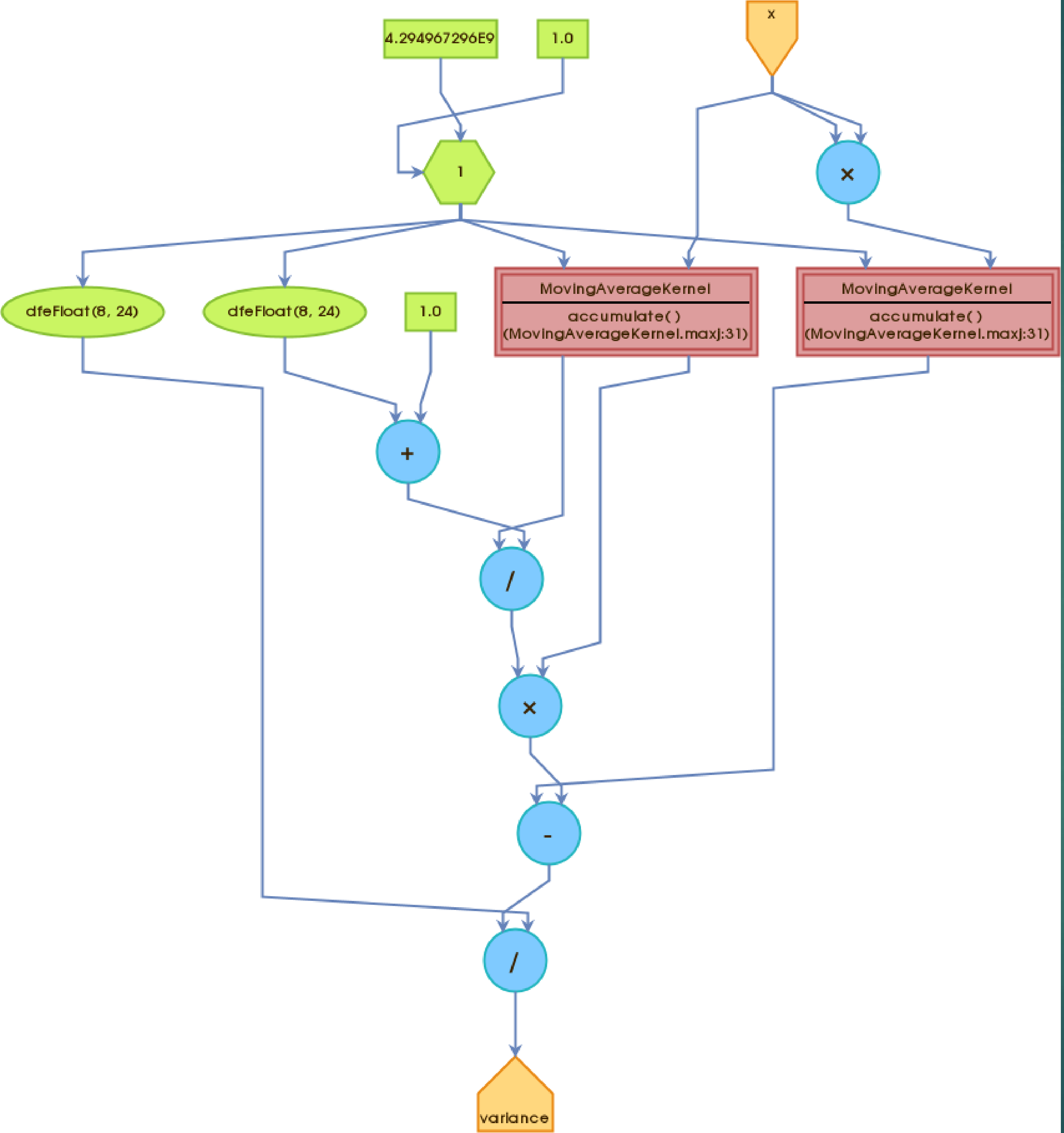
```
DFEType TYPE = dfeFloat(8, 24);
int LOOP_LENGHT = 15;

DFEVar count = control.count.simpleCounter(32);
DFEVar x = io.input("x", TYPE);

DFEVar subSum = constant.var(TYPE, 0.0);
for(int i = 0; i < LOOP_LENGHT; i++) {
        subSum = subSum + stream.offset(x, -i);
}

DFEVar carriedSum = TYPE.newInstance(this);
DFEVar temp = (count < LOOP_LENGHT) ? 0.0 :
carriedSum;
DFEVar sum = subSum + temp;
carriedSum <== stream.offset(sum, -LOOP_LENGHT);

io.output("sum", sum, TYPE);
```

Expanding Variance's Expression

- The variance depends on the mean.
- Expanding the variance to avoid reading twice the values.
- Two accumulators are enough to implement the last expression.

$$\mathrm{Var}(X) = \mathrm{E}\left[(X-\mu)^2\right].$$

$$\begin{aligned}\mathrm{Var}(X) &= \mathrm{E}\left[(X-\mathrm{E}[X])^2\right] \\ &= \mathrm{E}\left[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2\right] \\ &= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X]\,\mathrm{E}[X] + \mathrm{E}[X]^2 \\ &= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2\end{aligned}$$



Variance Kernel

- The accelerator reads every clock the input.
- The accumulator x and x^2 update their states.
- The accelerator outputs the variance up to that point.

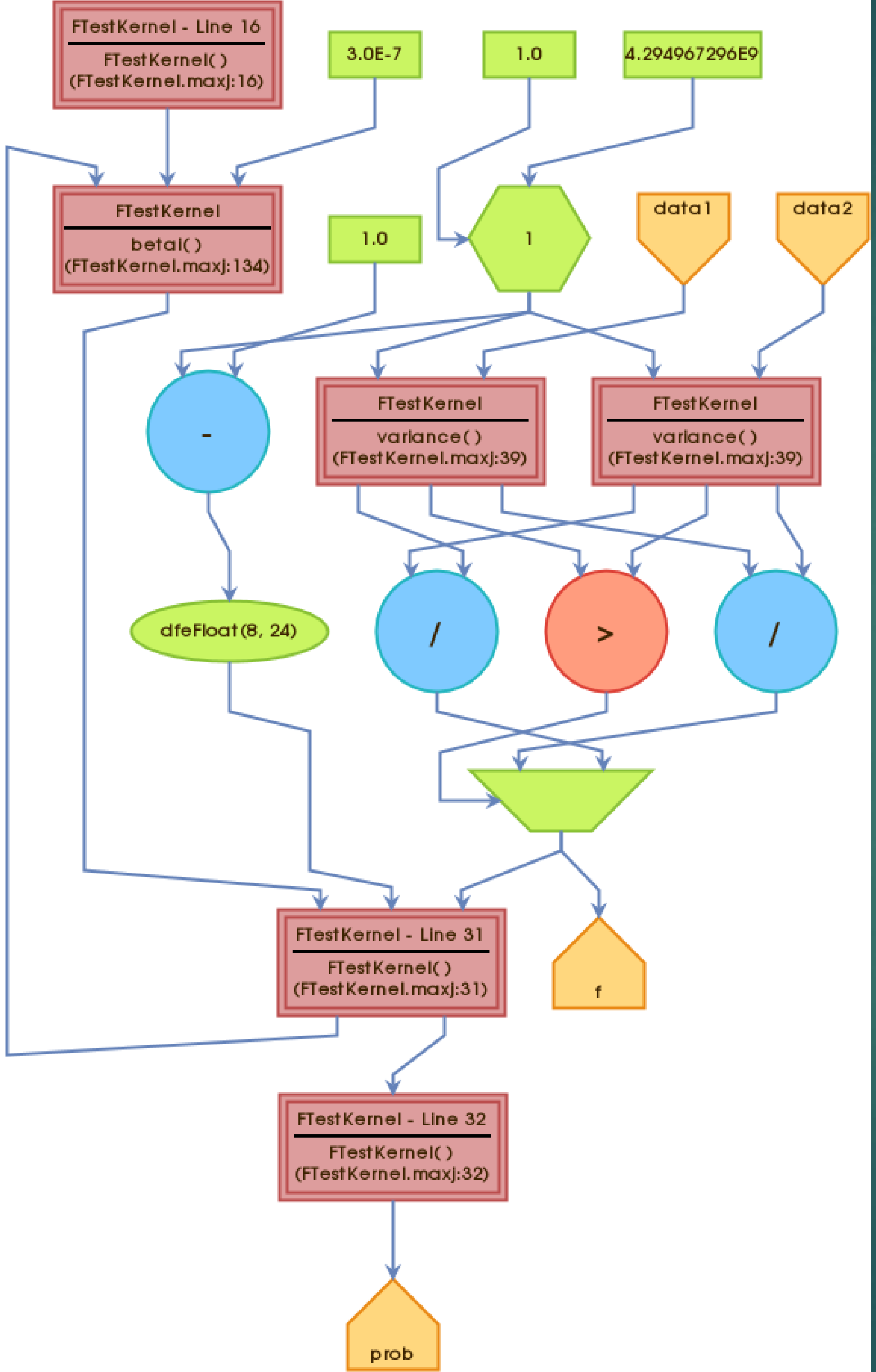
```
DFEType TYPE = dfeFloat(8, 24);

DFEVar count = control.count.simpleCounter(32);
DFEVar x = io.input("x", TYPE);

DFEVar sum = accumulate(x, count);
DFEVar squaredSum = accumulate(x * x, count);
DFEVar mean = sum / (count.cast(TYPE) + 1);

DFEVar variance = (squaredSum - mean * sum) / count.cast(TYPE);

io.output("variance", variance, TYPE);
```

F-Test

- x follows an F-distribution with n - 1 and m - 1 degrees of freedom.
- F is the cumulative distribution function of x.
- I is the regularized incomplete beta function.

$$x = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}{\frac{1}{m-1}\sum_{i=1}^{m}(Y_i - \overline{Y})^2}$$

$$F(x; d_1 = n-1; d_2 = m-1) = I_{\frac{d_1 x}{d_1 x + d_2}}\left(\frac{d_1}{2}, \frac{d_2}{2}\right)$$



F-Test Kernel

```
DFEType TYPE = dfeFloat(8, 24);
DFEVar count = control.count.simpleCounter(32);

DFEVar data1 = io.input("data1", TYPE);
DFEVar data2 = io.input("data2", TYPE);
DFEVar var1 = variance(data1, count);
DFEVar var2 = variance(data2, count);

DFEVar f = (var1 > var2) ? var1 / var2 : var2 / var1;
DFEVar df = (count - 1).cast(TYPE);
DFEVar prob = 2.0 * betai(0.5 * df, 0.5 * df, df / (df + df * f));
prob = (prob > 1.0) ? 2.0 - prob : prob;

io.output("f", f, TYPE);
io.output("prob", prob, TYPE);
```

Calling the F-Test Kernel from the Host

```
int main() {
  int n = 32;
  float data1[n], data2[n];
  for (int i = 0; i < n; ++i) {
    data1[i] = rand() / ((float)RAND_MAX + 1);
    data2[i] = rand() / ((float)RAND_MAX + 1);
    printf("%f %f\n", data1[i], data2[i]);
  }

  float f_result[n], prob_result[n];
  FTest(n, data1, data2, f_result, prob_result);
  printf("f=%f prob=%f\n", f_result[n-1], prob_result[n-1]);
  return 0;
}
```