CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 5 Memory & Addition
Sep 8, 2016

More Memory Fun

```
const a = rand(Int64,100_000_000)
const b = rand(Int64,100_000_000)
const c = rand(Int64,100_000_000)
const d = rand(Int64,100_000_000)
```

```
normal() = a .* b + c .* d + a
```

To compute normal()

```
temp1 = a .* b
temp2 = c .* d
r = temp1 + temp2 + a
```

.* is element by element multiplication

```
s = a .* b
s[1] = a[1] * b[1]
s[2] = a[2] * b[2]
```

Need two 100_000_000 temp vectors
Scan a twice

```
@time normal()
```

```
 10.977079 seconds (34 allocations: 2.980 GB, 12.68% gc time)
```

```
const a = rand(Int64,100_000_000)
const b = rand(Int64,100_000_000)
const c = rand(Int64,100_000_000)
const d = rand(Int64,100_000_000)
```

```
@time normal

 10.977079 seconds (34 allocations: 2.980 GB,
```

```
asloop() = begin
  n = length(a)
  r = zeros(n)
  for i = 1 : n
    r[i] = a[i] * b[i] + c[i] * d[i] + a[i]
  end
  r
end
```

```
@time asloop()

1.165048 seconds (6 allocations: 762.940 MB, 8.86% gc time)
```

devectorize

Devectorize will convert high level array operations into loops for you

```
Pkg.add("Devectorize")
using Devectorize
```

```
faster() = @devec r = a .* b + c .* d + a
```

@time normal()	
@time faster()	
@time asloop()	

Addition Fun

Addition Fun

```
tenth = fill(Float16(0.1),10)

sum(tenth)                          # 0.99975586f0



smalllarge = [100_000_000.0 1.0e-8 -100_000_000.0]

sum(smalllarge)                     # 0.0
```

Addition Fun

Issues

- Floats can not be represented exactly
- Errors can accumulate
- Large float + really small float loses lower order bits

Solutions

- Use more precision
- Use smarter addtion algorithms

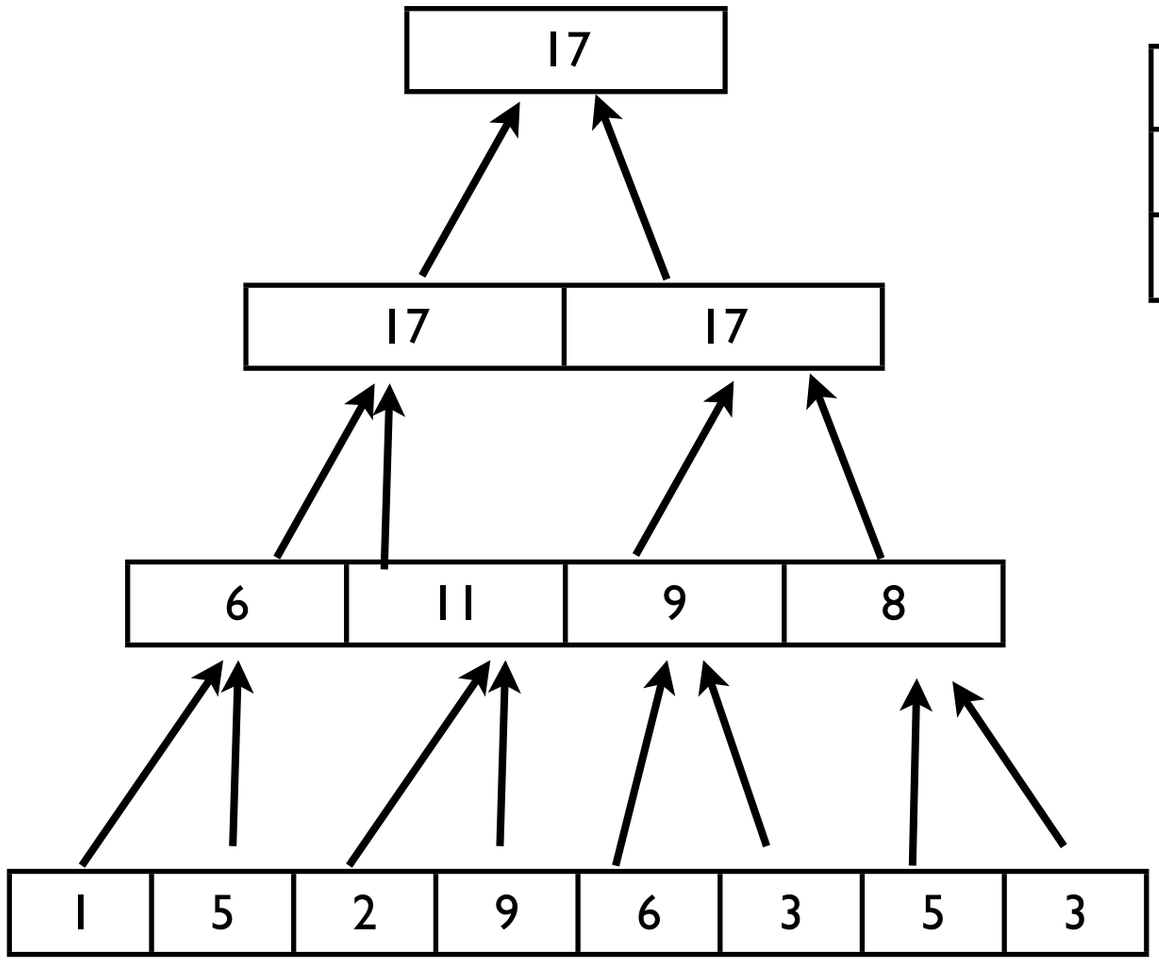
Using More Precision

```
tenth = fill(Float16(0.1),10)          sum(tenth)          # 0.99975586f0

tenth = fill(Float32(0.1),10)          sum(tenth)          # 1.0000001f0

tenth = fill(Float64(0.1),10)          sum(tenth)          # 0.9999999999999999

tenth = fill(BigFloat(0.1),10)         sum(tenth)          #
1.000000000000000055111512312578270211815834045410156250000000000000000000000
```

```
tenth = fill(TypeX(0.1),10_000_000)
@time sum(tenth)
```

TypeX	Time - Seconds
BigFloat	1.338
Float64	0.0179
Float32	0.0043
Float16	0.747

Use Smarter Algorithm - Pairwise Sum



	Error Growth Rate
Normal Sum	O(εn)
Pairwise Sum	O(εlog(n))

ε = machine precision

n = number of floats to add

NumPy & Julia default sum - pairwise

When n < k use normal sum

Pairwise Sum verses Linear Sum

```
function linear_sum(a)
  sum = 0.0
  for k in a
    sum += k
  end
  sum
end
```

```
tenth = fill(0.1,10_000_000)
```

	Result	Time - Seconds
sum(tenth)	999999.999999997	0.008
linear_sum(tenth)	999999.999838975	0.017
reduce(+, tenth)	999999.999999997	0.007

Using More Precision

smalllarge = TypeX[100_000_000.0 1.0e-8 -100_000_000.0]

sum(smalllarge)

TypeX	Result	Time - Seconds
Float32	0.0f0	0.000_003
Float64	1.4901161193847700000E-08	0.000_004
BigFloat	1.0000000000000000000E-08	0.017_866

Use Smarter Algorithm - Kahan Summation

```
function KahanSum(input)
    sum = 0.0
    c = 0.0                  // A running compensation for lost low-order bits.
    for k in input do
        y = k - c
        t = sum + y          // sum is big, y small, so low-order digits of y are lost.
        c = (t - sum) - y    // (t - sum) cancels the high-order part of y;
                             //  subtracting y recovers negative (low part of y)
        sum = t              // Beware overly-aggressive optimizing compilers!
    end
    return sum
end
```

	Error Growth Rate
Normal Sum	O(εn)
Pairwise Sum	O(εlog(n))
Kahan Sum	O(ε)

ε = machine precision

n = number of floats to add

Kahan Summation

smalllarge = [100_000_000.0 1.0e-8 -100_000_000.0]

	Result	Time
sum(smalllarge)	0.0f0	0.000_003
sum_kbn(smalllarge)	1.00E-08	0.000_003

tenth = fill(0.1,10_000_000)

	Result	Time
sum( )	999999.9999999970	0.007_9
sum_kbn( )	1.00E+06	0.032_2