Husky: A Functional Library for C++

Ayo Fapohunda (oaf2119@columbia.edu)

Larissa Passos (ln2307@columbia.edu)

Vinicius Cousseau (vd2299@columbia.edu)

Outline

- Functional programming context
- Previous approaches in imperative languages (C++)
- Our Design
- Comparisons
- Our results

Functional Programming

- What is functional programming?
  - Programming paradigm
  - Mathematical functions
  - Avoids side-effects
  - Abstractions
- Contrast to imperative programming
  - Subroutines

Quicksort

```
// lo is the index of the leftmost element of the subarray
// hi is the index of the rightmost element of the subarray (inclusive)
partition(A, lo, hi)
   pivotIndex := choosePivot(A, lo, hi)
   pivotValue := A[pivotIndex]
   // put the chosen pivot at A[hi]
   swap A[pivotIndex] and A[hi]
   storeIndex := lo
   // Compare remaining array elements against pivotValue = A[hi]
   for i from lo to hi-1, inclusive
       if A[i] <= pivotValue
           swap A[i] and A[storeIndex]
           storeIndex := storeIndex + 1
   swap A[storeIndex] and A[hi]  // Move pivot to its final place
   return storeIndex
```

```
quicksort(A, lo, hi):
  if lo < hi:
    p := partition(A, lo, hi)
    quicksort(A, lo, p - 1)
    quicksort(A, p + 1, hi)
```

“Quicksort”

```
quicksort [] = []
quicksort (p:xs) = (quicksort lesser) ++ [p] ++
                   (quicksort greater)
                    where
                        lesser  = filter (< p)  xs
                        greater = filter (>= p) xs
```

“Quicksort”

```
qs (a:as) = qs [x| x <- as, x <= a] ++ [a]
            ++ qs [ x | x <- as, x > a]
```

Comparison Targets

- Existing libraries: FC++ (2000) and FTL (~2014)
- Other languages: Haskell, Python
- Criteria

Husky Design

- ~50 Functions based on Haskell Prelude
- General structure
- Tests, tests, tests...

Husky Design

```cpp
#include <iostream>
#include <string>
#include "husky.h"

using namespace husky;
using namespace std;

auto caps = [](char c) { return (c >= 65 && c <= 90); };

int main() {
    string str = "HelloUSweetKoalaYou";
    string s = filter(str, caps);
    cout << s << endl;
    return 0;
}
```

HUSKY

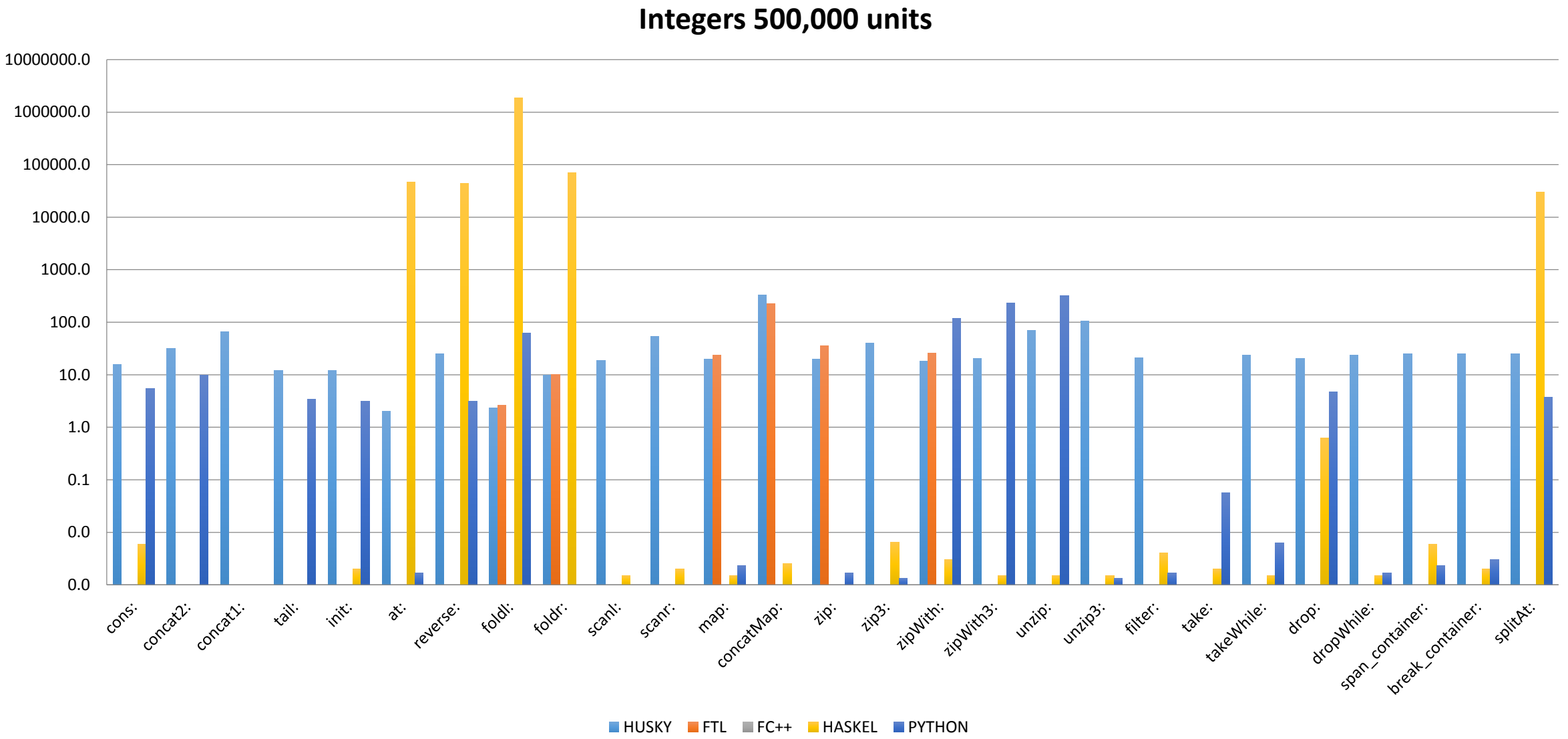
Testing Suite

- OS X 10.10.3
  - 2.3 GHz intel i7 quad-core
  - 16gb RAM 1600Mhz DDR3
  - Clang++
- Input vectors of
    - int
    - std::string
    - Record { int, std::string }
- Average of 3 iterations for each input vector, 100k – 500k

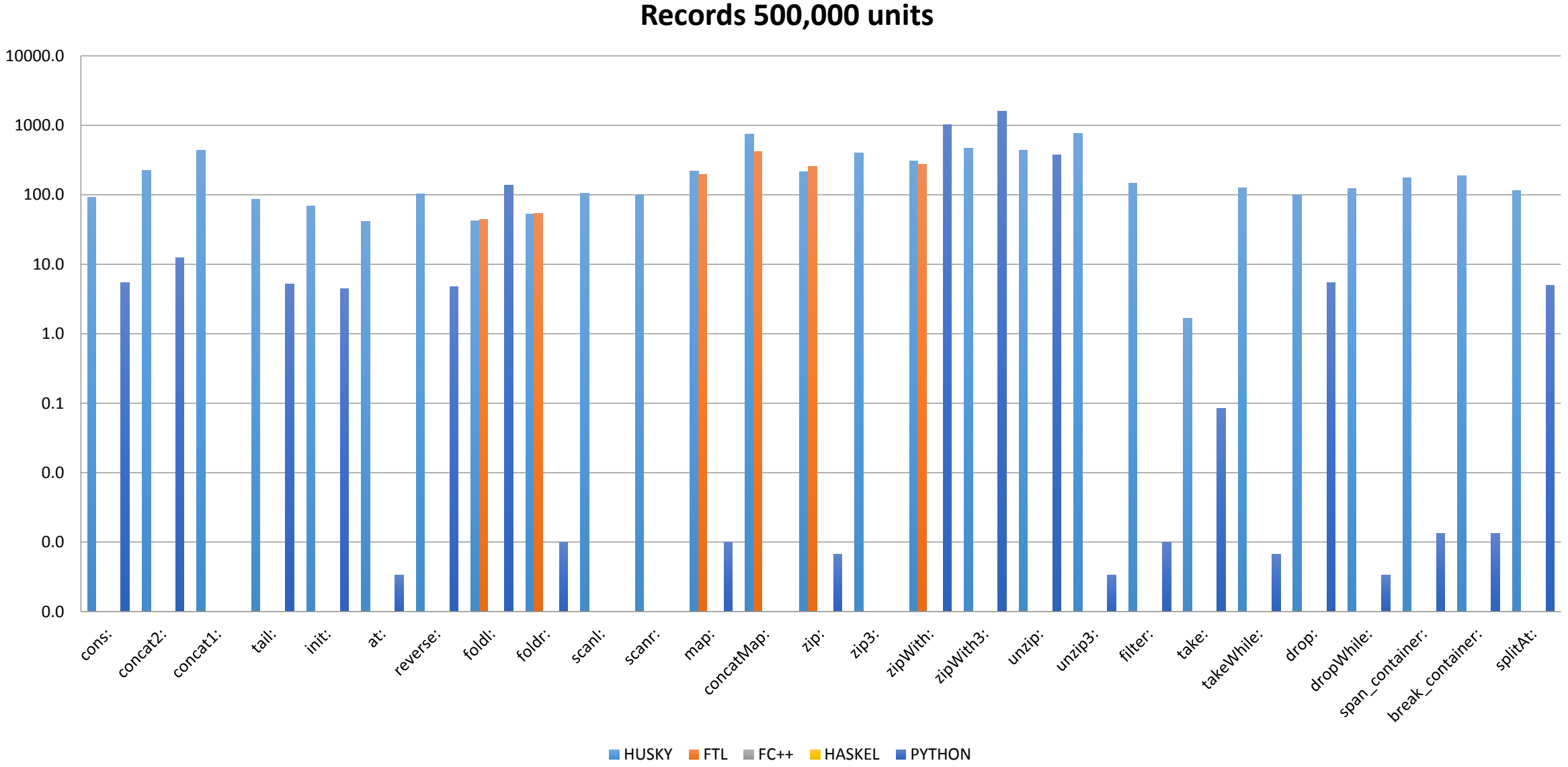
Additional Considerations

- Less Functions
- Timing in Haskell
- Lack of proper documentation/tutorials (FC++ and FTL)

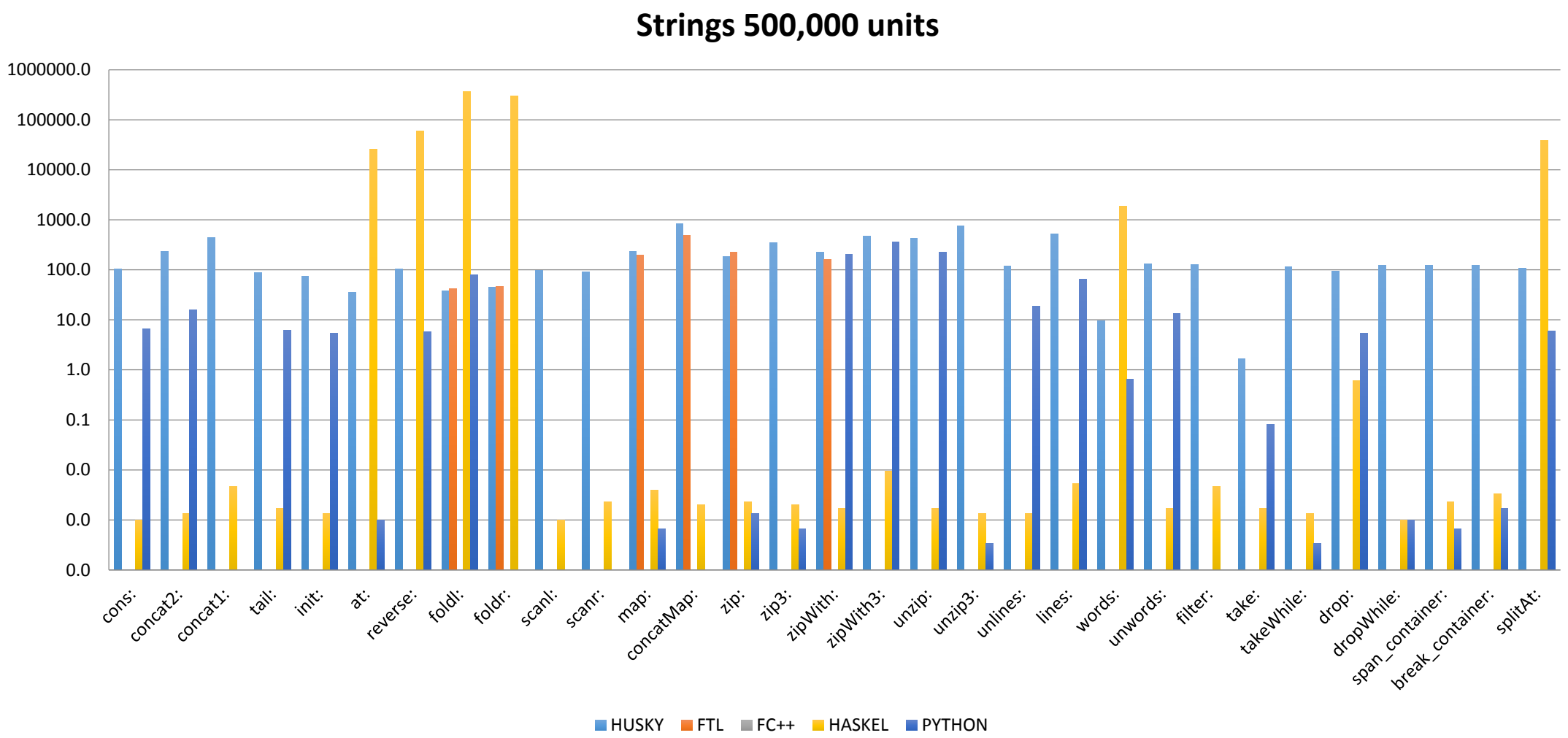
Results


Integers 500,000 units
10000000.0
1000000.0
100000.0
10000.0
1000.0
100.0
10.0
1.0
0.1
0.0
0.0
cons:
concat2:
concat1:
tail:
init:
at:
reverse:
foldl:
foldr:
scanl:
scanr:
map:
concatMap:
zip:
zip3:
zipWith:
zipWith3:
unzip:
unzip3:
filter:
take:
takeWhile:
drop:
dropWhile:
span_container:
break_container:
splitAt:
HUSKY
FTL
FC++
HASKEL
PYTHON

Results


Records 500,000 units
10000.0
1000.0
100.0
10.0
1.0
0.1
0.0
0.0
0.0
cons:
concat2:
concat1:
tail:
init:
at:
reverse:
foldl:
foldr:
scanl:
scanr:
map:
concatMap:
zip:
zip3:
zipWith:
zipWith3:
unzip:
unzip3:
filter:
take:
takeWhile:
drop:
dropWhile:
span_container:
break_container:
splitAt:
HUSKY
FTL
FC++
HASKEL
PYTHON

Results


Strings 500,000 units
1000000.0
100000.0
10000.0
1000.0
100.0
10.0
1.0
0.1
0.0
0.0
0.0
cons:
concat2:
concat1:
tail:
init:
at:
reverse:
foldl:
foldr:
scanl:
scanr:
map:
concatMap:
zip:
zip3:
zipWith:
zipWith3:
unzip:
unzip3:
unlines:
lines:
words:
unwords:
filter:
take:
takeWhile:
drop:
dropWhile:
span_container:
break_container:
splitAt:
HUSKY
FTL
FC++
HASKEL
PYTHON

Additional Testing Suite

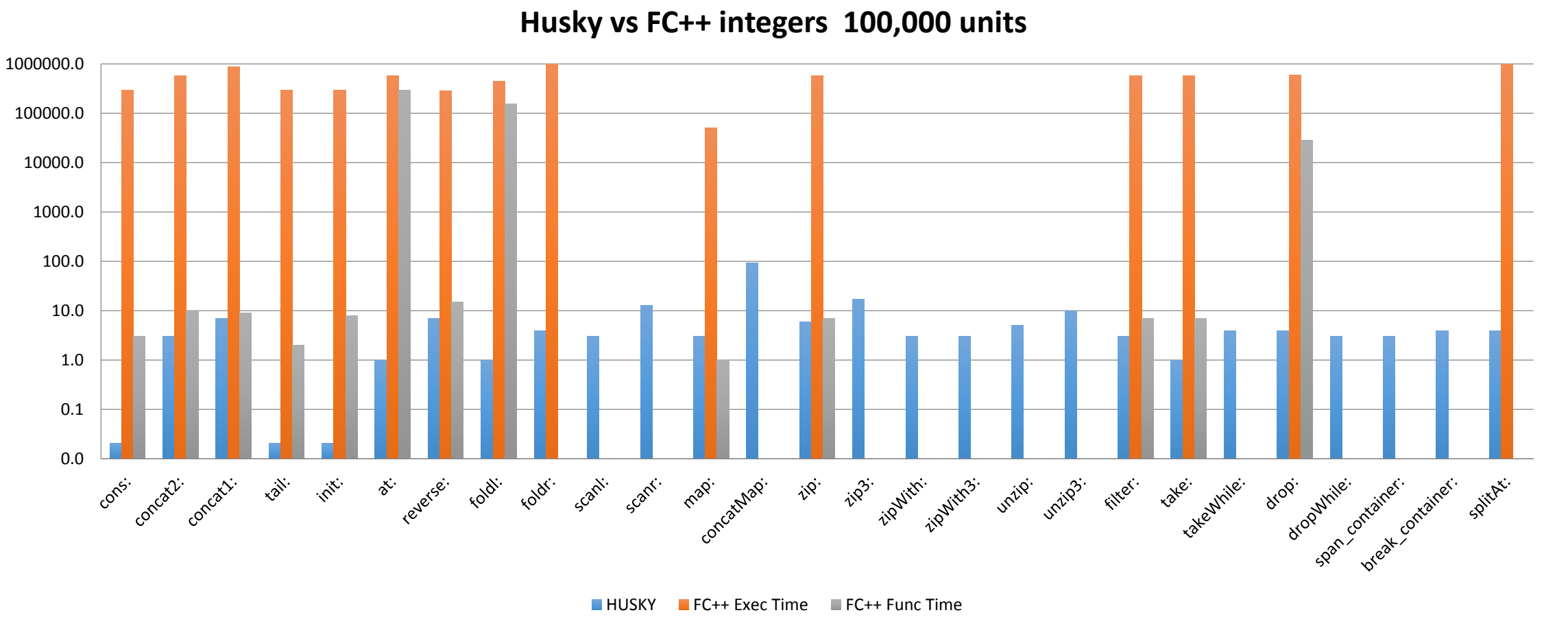
- Windows 8.1 64-bit
  - 16 Gb RAM
  - 2.5 GHz i7
  - g++
- Input vectors of ints, size 100k

Test Example

// Long lists create long recursions of destructors that blow the
// stack. So we have an iterative destructor. It is quite tricky to
// get right. The danger is that, when "bypassing" a node to be
// unlinked and destructed, that node's 'next' pointer is, in fact, a
// List object, whose destructor will be called. As a result, as you
// bypass a node, you need to see if its refC is down to 1, and if
// so, mutate its next pointer so that when its destructor is called,
// it won't cause a recursive cascade.

Long lists ~ 130k elements

Results


Husky vs FC++ integers 100,000 units
1000000.0
100000.0
10000.0
1000.0
100.0
10.0
1.0
0.1
0.0
cons:
concat2:
concat1:
tail:
init:
at:
reverse:
foldl:
foldr:
scanl:
scanr:
map:
concatMap:
zip:
zip3:
zipWith:
zipWith3:
unzip:
unzip3:
filter:
take:
takeWhile:
drop:
dropWhile:
span_container:
break_container:
splitAt:
HUSKY
FC++ Exec Time
FC++ Func Time

Summation

- Comprehensive higher order functional library
- Extension of STL
- More intuitive way to code
- Performs better than our main benchmarks (FC++ & FTL)
- Far more and far better documentation/tutorials (FC++ & FTL)

Acknowledgements

- FTL : https://github.com/beark/ftl

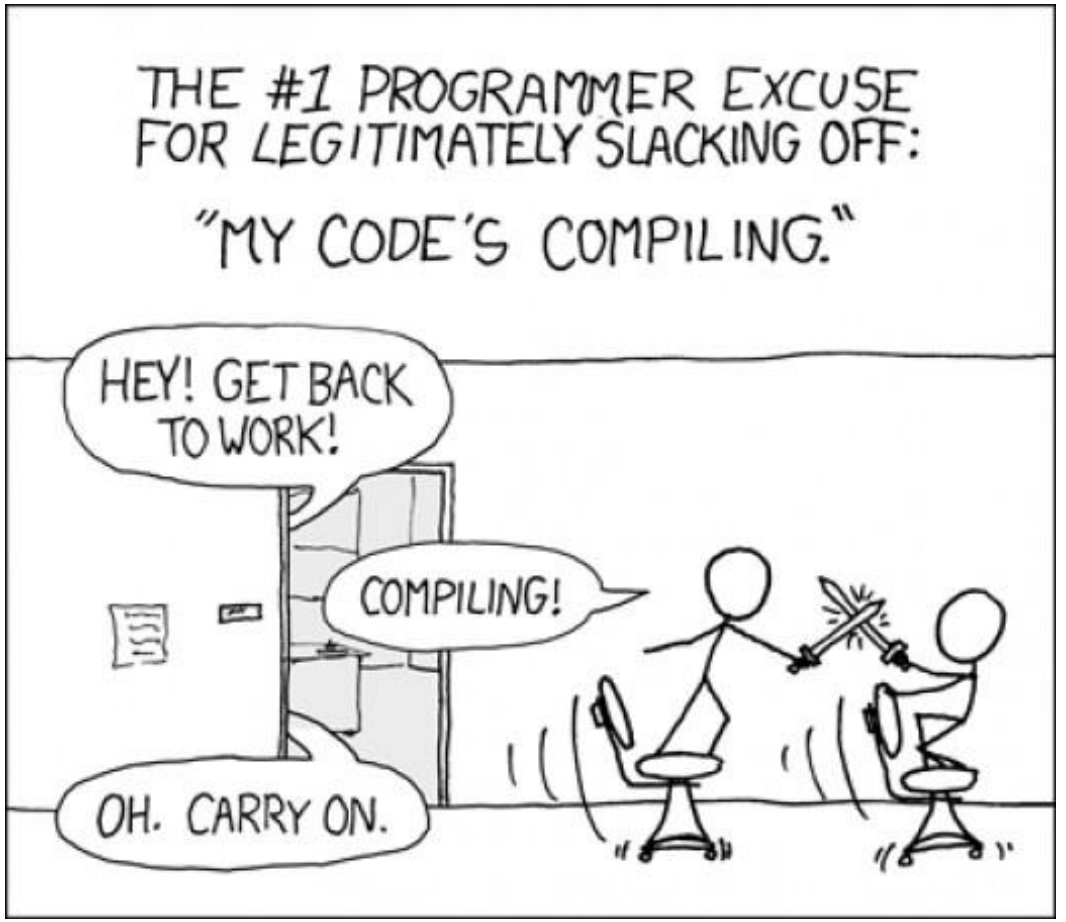
- FC++ : http://cgi.di.uoa.gr/~smaragd/fc++/

Final Considerations

- C++14 extending even more!
- (1.2) Currying, Lazy Evaluation ?

Final Considerations

- ln2307@columbia.edu
- oaf2119@columbia.edu
- vd2299@columbia.edu



References

- Husky will be available at https://github.com/larissapassos/Husky

- More about functional programming:
  - https://wiki.haskell.org/Introduction
  - http://learnyouahaskell.com/