Husky: A Functional Library for C++

Ayo Fapohunda (oaf2119@columbia.edu)

Larissa Passos (ln2307@columbia.edu)

Vinicius Cousseau (vd2299@columbia.edu)

Abstract

Project object was to write higher order functions for C++11/14 that extend the current STL, in order to enable better functional programming, showing how it compares to other libraries/languages.

Show that C++11/14 provides much better resources for functional-style programming.

Time needed:

Q&A:

(Slide will be hidden)

Outline

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

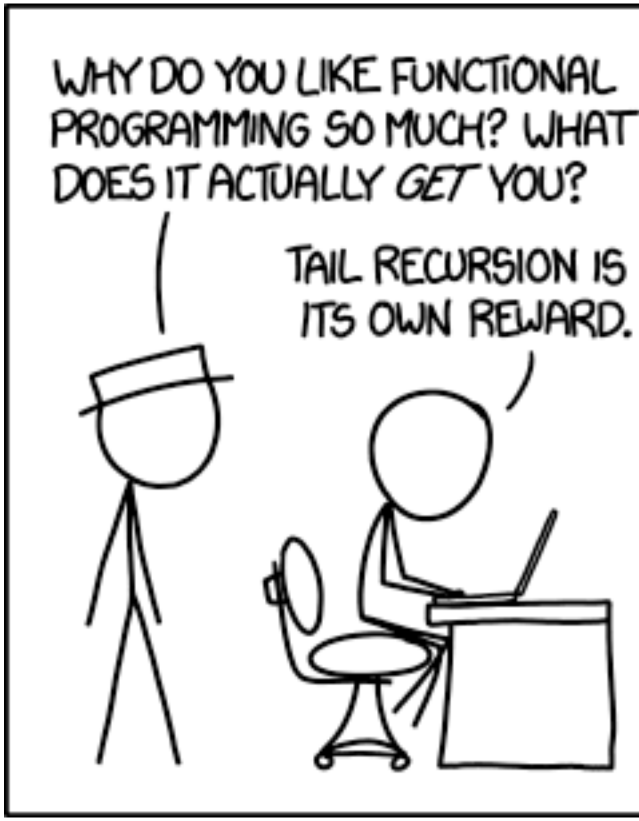
Functional Programming

What is functional programming?

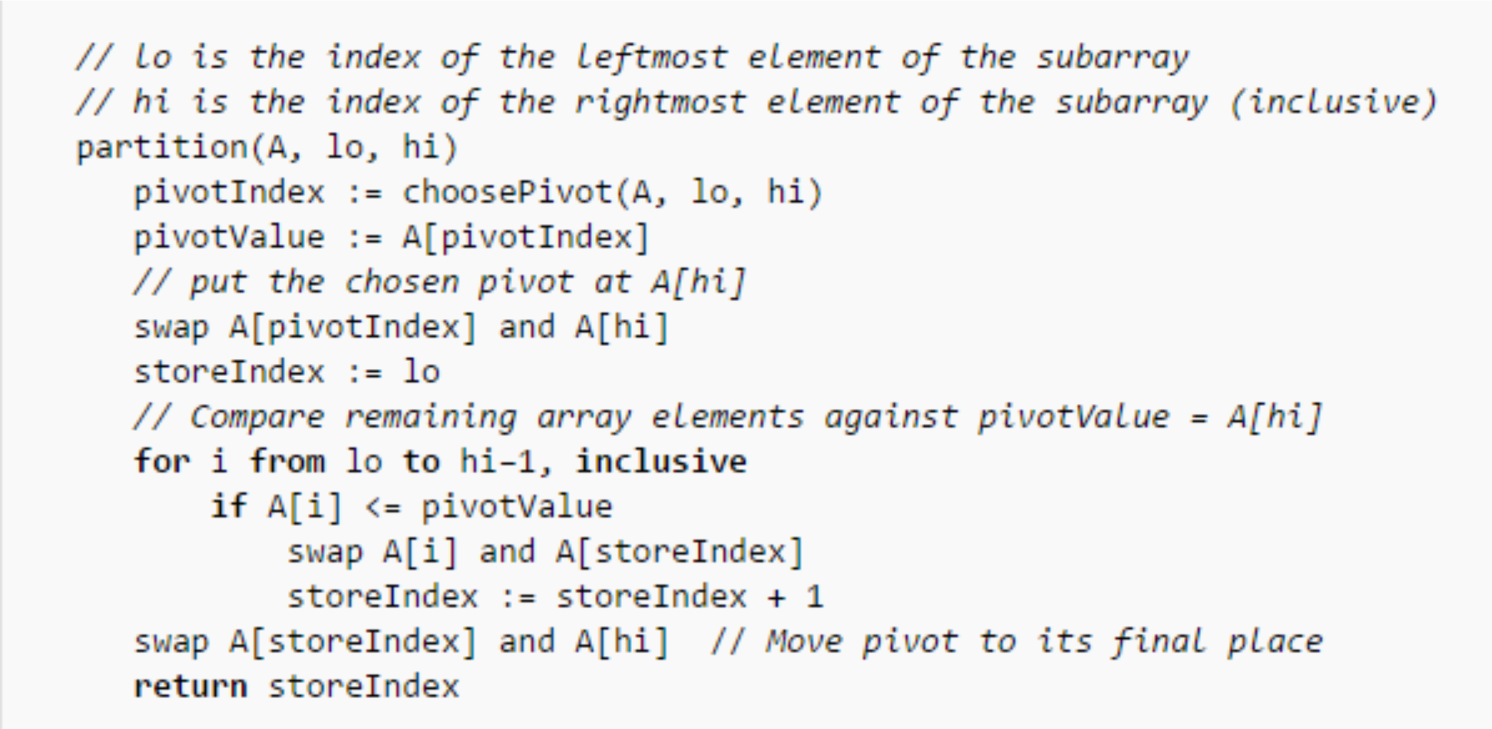
- Programming paradigm
- Mathematical functions
- Avoids side-effects
- Abstractions

Contrast to imperative programming

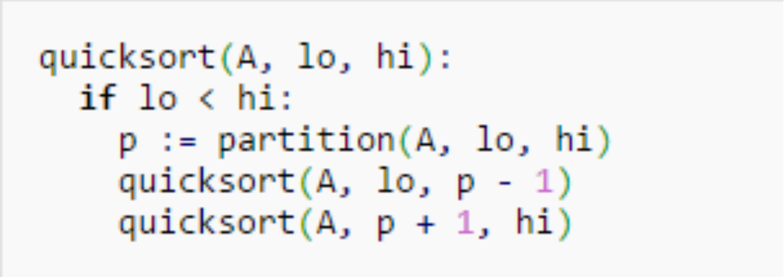
- Subroutines


WHY DO YOU LIKE FUNCTIONAL PROGRAMMING SO MUCH? WHAT DOES IT ACTUALLY *GET* YOU?
TAIL RECURSION IS ITS OWN REWARD.

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

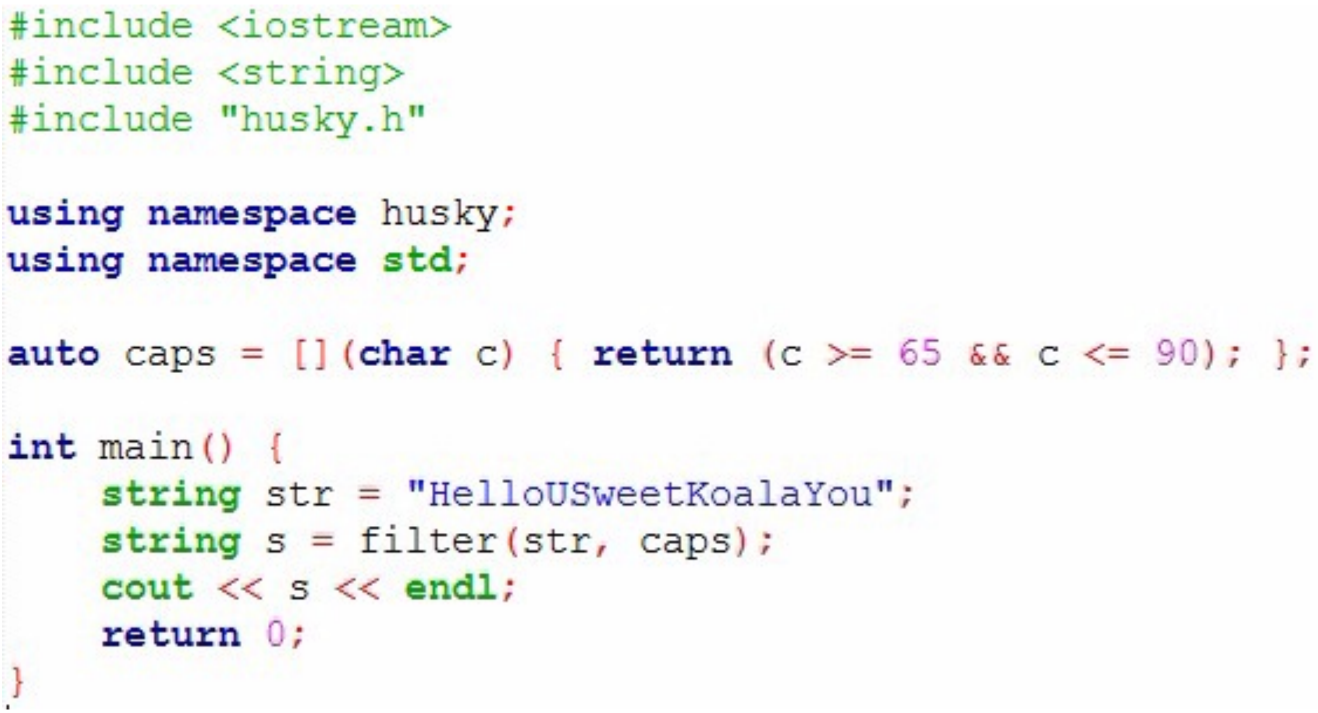
Husky Design

~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

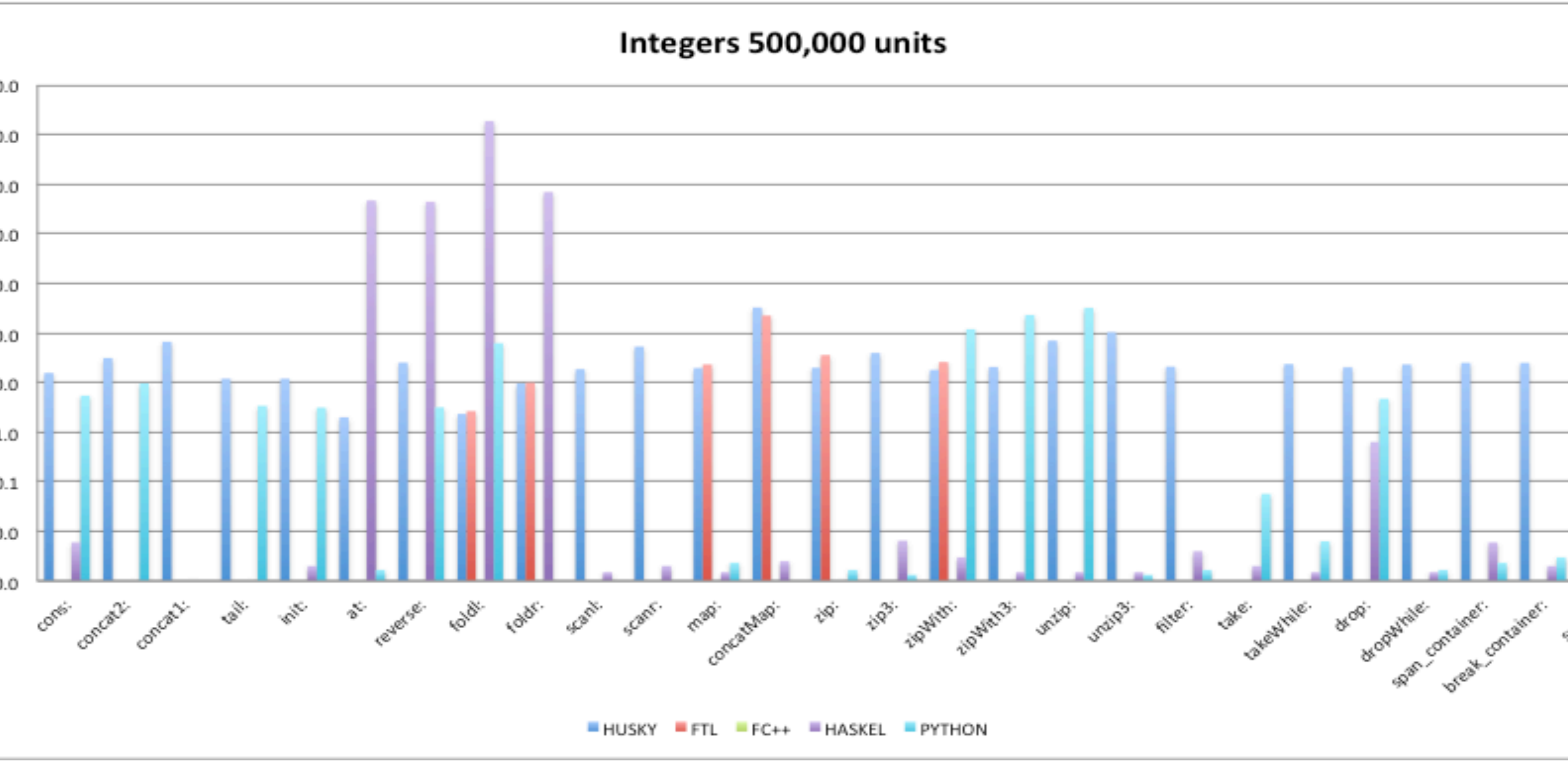
Additional Considerations

Less Functions

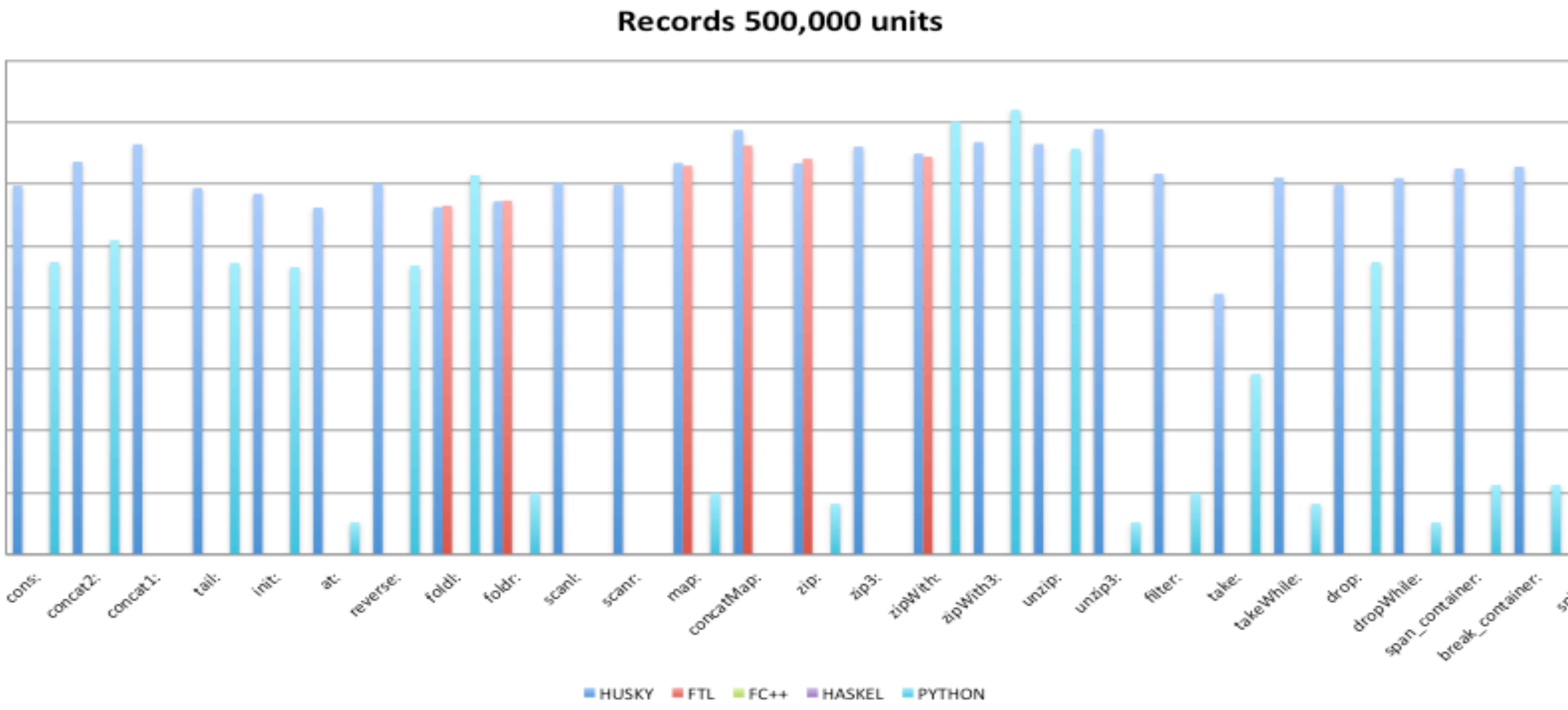
Timing in Haskell

Lack of proper documentation/tutorials (FC++ and FTL)

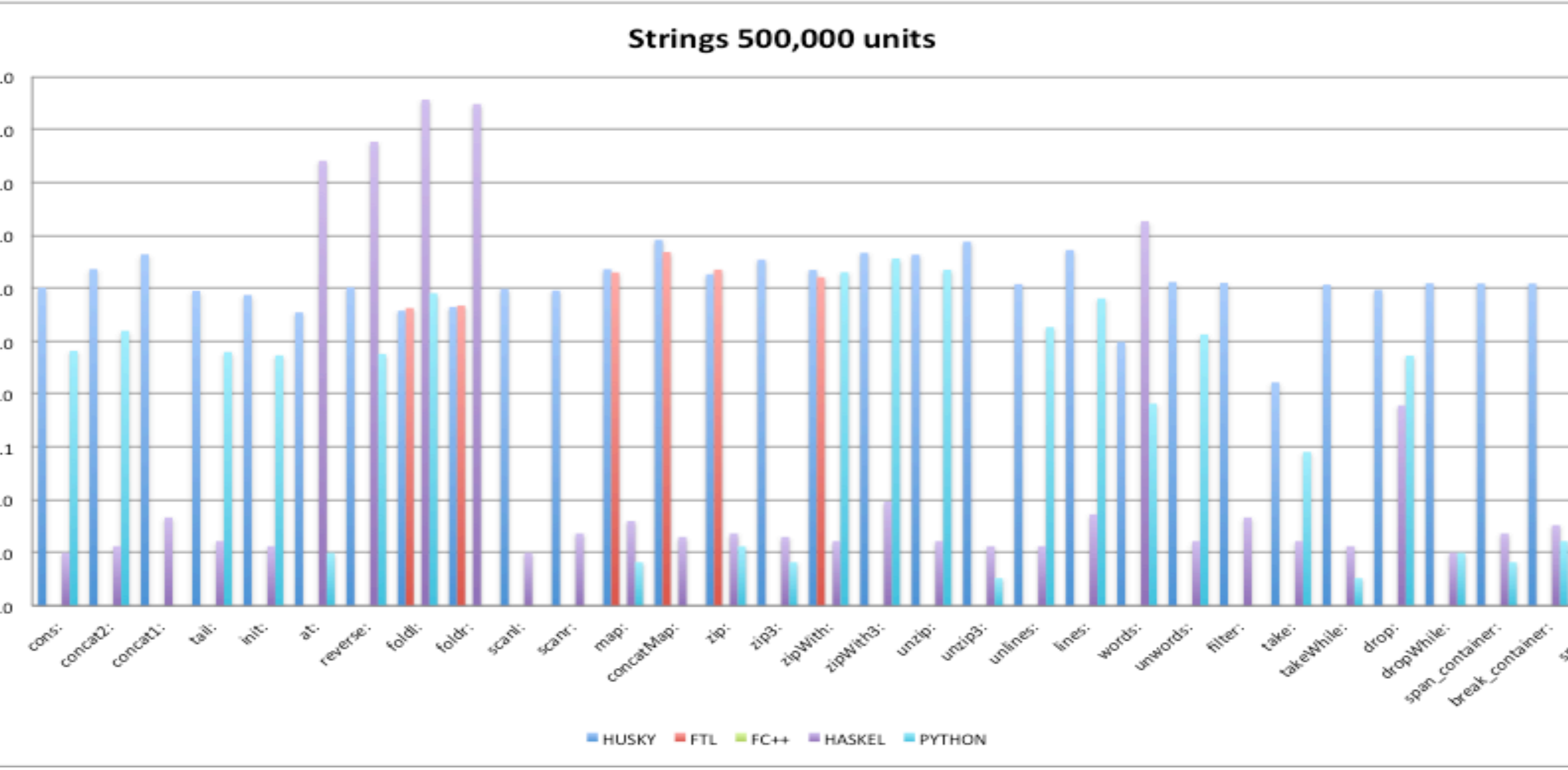
Results


Integers 500,000 units
).0
).0
).0
).0
).0
).0
).0
1.0
).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:
HUSKY
FTL
FC++
HASKEL
PYTHON

Results


Records 500,000 units
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:
HUSKY
FTL
FC++
HASKEL
PYTHON

Results


Strings 500,000 units
.0
.0
.0
.0
.0
.0
.0
.1
.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:
HUSKY
FTL
FC++
HASKEL
PYTHON

Additional Testing Suite

Windows 8.1 64-bit

- 16 Gb RAM
- 2.5 GHz i7
- g++

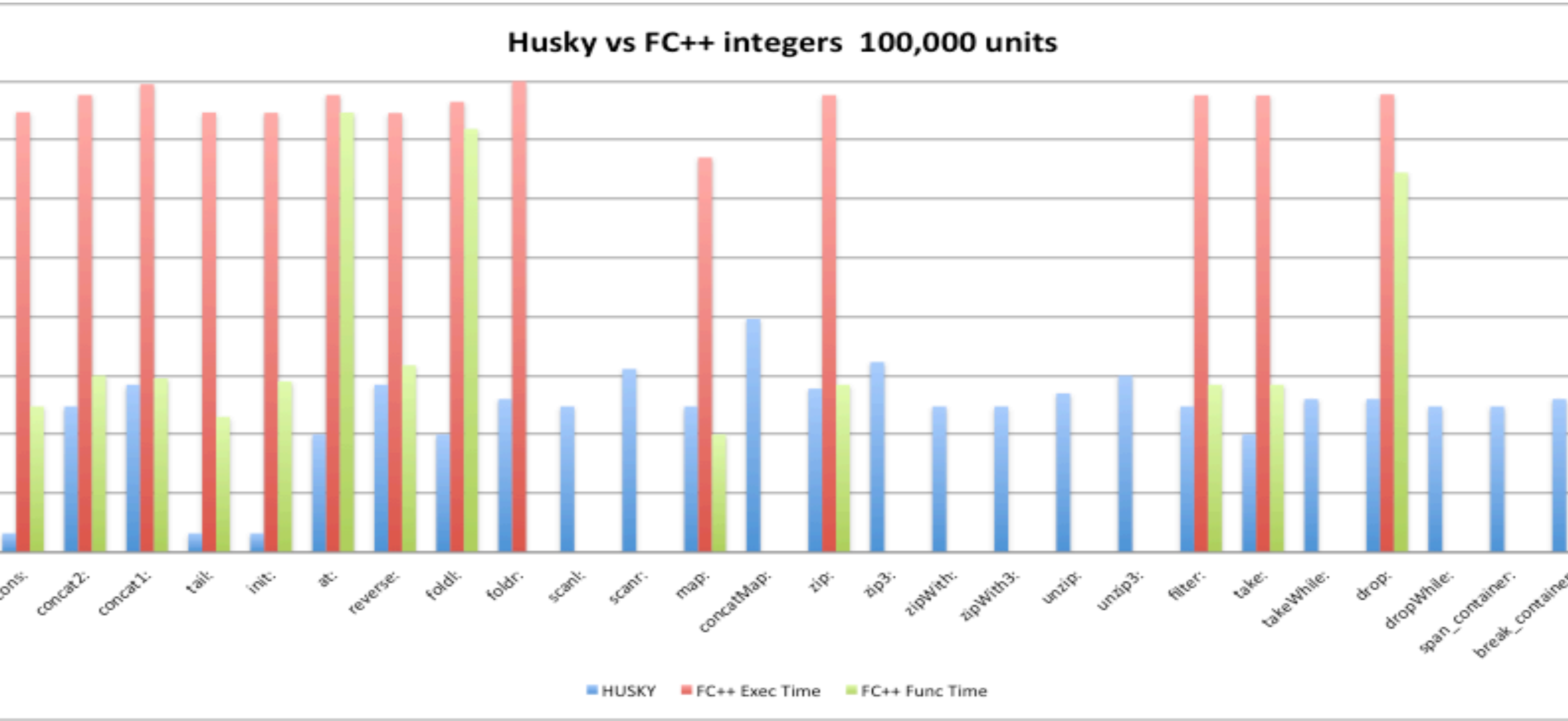
Input vectors of ints, size 100k

Case Study

```
/ 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
ons:
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_containe
HUSKY
FC++ Exec Time
FC++ Func Time

Results

Results for Integers

			Husky
N = 100000	HUSKY	FC++ Func Time	% faster
cons:	0.02	3.0	14186%
concat2:	3.00	10.0	233%
concat1:	7.00	9.0	29%
tail:	0.02	2.0	9424%
init:	0.02	8.0	37995%
at:	1.00	292030.0	29202900%
reverse:	7.00	15.0	114%
foldl:	1.00	154705.0	15470400%
map:	3.00	1.0	-67%
zip:	6.00	7.0	17%
filter:	3.00	7.0	133%
take:	1.00	7.0	600%
drop:	4.00	28018.0	700350%

Results

l = 500000	HUSKY	FTL	Husky % faster
oldl:	2.3	2.7	14%
oldr:	9.7	10.0	3%
nap:	19.7	23.3	19%
oncatMap:	326.7	225.0	-31%
ip:	20.0	36.0	80%
ipWith:	18.0	26.0	44%

Strings 500,000 elements

l = 500000	HUSKY	FTL	Husky % faster
oldl:	37.7	42.3	12%
oldr:	44.3	47.0	6%
nap:	230.0	198.7	-14%
oncatMap:	822.7	483.0	-41%
ip:	184.0	225.0	22%
ipWith:	222.0	161.7	-27%

Records 500,000 elements

N = 500000	HUSKY	FTL	Husky % faster
foldl:	42.0	44.3	6%
foldr:	52.3	53.7	3%
map:	219.0	197.3	-10%
concatMap:	743.3	419.0	-44%
zip:	216.0	255.3	18%
zipWith:	308.7	274.3	-11%

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/