# Comparing Android Runtime with native: Fast Fourier Transform on Android

André Danielsson

*anddani@kth.se*

Royal Institute of Technology
Computer Science and Communication

May 25, 2017

## Purpose of Thesis

- Why is this work important?
- Who will benefit from it?

## Research Question

*Is there a significant performance difference between implementations of a Fast Fourier Transform (FFT) in native code, compiled by Clang, and Dalvik bytecode, compiled by Android Runtime, on Android?*

## Android development

- Android Software Development Kit (SDK)

## Android development

- Android Software Development Kit (SDK)
- Android Runtime

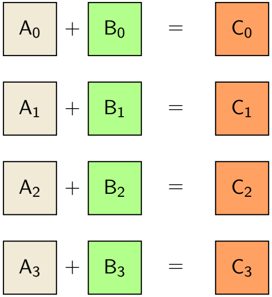
## Android development

- Android Software Development Kit (SDK)
- Android Runtime
- Android Native Development Kit (NDK)

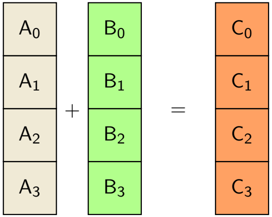
## Android development

- Android Software Development Kit (SDK)
- Android Runtime
- Android Native Development Kit (NDK)
- Java Native Interface (JNI)

# Vectorization



(a) Four separate instructions



(b) One instruction with SIMD

- SSE
- NEON

## Discrete Fourier Transform (DFT)

- DFT: *Time* $\Rightarrow$ *Frequency*
- Decomposition of a signal
- Example uses:
  - Audio visualization
  - Speech recognition
  - Compression
- Fast Fourier Transform (FFT)

## Experiments

1. Cost of using JNI
2. Comparison of smaller FFT libraries
3. Native optimization with NEON

## Implementation

- **Java**
    - Princeton Recursive
    - Princeton Iterative
    - Columbia Iterative

## Implementation

- **Java**
  - Princeton Recursive
  - Princeton Iterative
  - Columbia Iterative
- **C++**
  - KISS (Keep It Simple Stupid) FFT
  - SSE Recursive FFT
  - SSE Iterative FFT

## Implementation

- Benchmark application
- Separate thread, one algorithm at a time
- Time measurements executed on a release build
- Memory measurements executed with an attached debugger

## JNI (Time µs)

| Block size | No params | Vector | Convert | Columbia |
|---|---|---|---|---|
| **16** | 1.7922 ± 0.1392 | 1.9333 ± 0.1223 | 2.6052 ± 0.1004 | 4.1058 ± 0.3042 |
| **32** | 1.6983 ± 0.0220 | 2.8130 ± 1.7924 | 2.6006 ± 0.0370 | 3.9109 ± 0.0535 |
| **64** | 1.6755 ± 0.0149 | 1.6344 ± 0.1809 | 2.6630 ± 0.0425 | 3.9296 ± 0.0566 |
| **128** | 1.9604 ± 0.4978 | 1.2349 ± 0.1262 | 1.9375 ± 0.0843 | 3.0823 ± 0.0892 |
| **256** | 1.7292 ± 0.0694 | 1.3276 ± 0.2589 | 1.8141 ± 0.0276 | 3.0958 ± 0.0441 |
| **512** | 1.6916 ± 0.0110 | 1.2567 ± 0.1227 | 2.2818 ± 0.7011 | 3.1656 ± 0.0457 |
| **1024** | 2.0228 ± 0.5684 | 1.3167 ± 0.1341 | 6.3756 ± 8.4676 | 3.2896 ± 0.1396 |
| **2048** | 1.7218 ± 0.0288 | 1.5416 ± 0.1405 | 1.9099 ± 0.0898 | 3.4844 ± 0.1113 |
| **4096** | 1.1411 ± 0.0404 | 1.4010 ± 0.0788 | 2.0062 ± 0.1562 | 3.8562 ± 0.3197 |
| **8192** | 1.1105 ± 0.0078 | 1.4818 ± 0.0759 | 2.3671 ± 0.1897 | 3.8474 ± 0.4784 |
| **16384** | 1.1183 ± 0.0280 | 1.7308 ± 0.1043 | 2.5833 ± 0.1737 | 4.9724 ± 0.8955 |
| **32768** | 1.1162 ± 0.0084 | 2.2099 ± 0.1880 | 3.2062 ± 0.2029 | 5.3719 ± 0.2875 |
| **65536** | 1.7463 ± 1.2217 | 4.7474 ± 3.1960 | 4.3198 ± 0.2926 | 6.8136 ± 0.2499 |
| **131072** | 1.1027 ± 0.0141 | 2.6375 ± 0.1531 | 5.7004 ± 0.2681 | 9.6912 ± 1.4337 |
| **262144** | 1.1006 ± 0.0118 | 3.3172 ± 0.1164 | 7.4630 ± 0.2309 | 10.2781 ± 0.2278 |

## JNI (Time µs)

| Block size | No params | Vector | Convert | Columbia |
|---|---|---|---|---|
| **16** | 1.7922 ± 0.1392 | 1.9333 ± 0.1223 | 2.6052 ± 0.1004 | 4.1058 ± 0.3042 |
| **32** | 1.6983 ± 0.0220 | 2.8130 ± 1.7924 | 2.6006 ± 0.0370 | 3.9109 ± 0.0535 |
| **64** | 1.6755 ± 0.0149 | 1.6344 ± 0.1809 | 2.6630 ± 0.0425 | 3.9296 ± 0.0566 |
| **128** | 1.9604 ± 0.4978 | 1.2349 ± 0.1262 | 1.9375 ± 0.0843 | 3.0823 ± 0.0892 |
| **256** | 1.7292 ± 0.0694 | 1.3276 ± 0.2589 | 1.8141 ± 0.0276 | 3.0958 ± 0.0441 |
| **512** | 1.6916 ± 0.0110 | 1.2567 ± 0.1227 | 2.2818 ± 0.7011 | 3.1656 ± 0.0457 |
| **1024** | 2.0228 ± 0.5684 | 1.3167 ± 0.1341 | $\boxed{6.3756 \pm 8.4676}$ | 3.2896 ± 0.1396 |
| **2048** | 1.7218 ± 0.0288 | 1.5416 ± 0.1405 | 1.9099 ± 0.0898 | 3.4844 ± 0.1113 |
| **4096** | 1.1411 ± 0.0404 | 1.4010 ± 0.0788 | 2.0062 ± 0.1562 | 3.8562 ± 0.3197 |
| **8192** | 1.1105 ± 0.0078 | 1.4818 ± 0.0759 | 2.3671 ± 0.1897 | 3.8474 ± 0.4784 |
| **16384** | 1.1183 ± 0.0280 | 1.7308 ± 0.1043 | 2.5833 ± 0.1737 | 4.9724 ± 0.8955 |
| **32768** | 1.1162 ± 0.0084 | 2.2099 ± 0.1880 | 3.2062 ± 0.2029 | 5.3719 ± 0.2875 |
| **65536** | 1.7463 ± 1.2217 | 4.7474 ± 3.1960 | 4.3198 ± 0.2926 | 6.8136 ± 0.2499 |
| **131072** | 1.1027 ± 0.0141 | 2.6375 ± 0.1531 | 5.7004 ± 0.2681 | 9.6912 ± 1.4337 |
| **262144** | 1.1006 ± 0.0118 | 3.3172 ± 0.1164 | 7.4630 ± 0.2309 | 10.2781 ± 0.2278 |

## JNI (Time µs)

| Block size | No params | Vector | Convert | Columbia |
|---|---|---|---|---|
| **16** | 1.7922 ± 0.1392 | 1.9333 ± 0.1223 | 2.6052 ± 0.1004 | 4.1058 ± 0.3042 |
| **32** | 1.6983 ± 0.0220 | 2.8130 ± 1.7924 | 2.6006 ± 0.0370 | 3.9109 ± 0.0535 |
| **64** | 1.6755 ± 0.0149 | 1.6344 ± 0.1809 | 2.6630 ± 0.0425 | 3.9296 ± 0.0566 |
| **128** | 1.9604 ± 0.4978 | 1.2349 ± 0.1262 | 1.9375 ± 0.0843 | 3.0823 ± 0.0892 |
| **256** | 1.7292 ± 0.0694 | 1.3276 ± 0.2589 | 1.8141 ± 0.0276 | 3.0958 ± 0.0441 |
| **512** | 1.6916 ± 0.0110 | 1.2567 ± 0.1227 | 2.2818 ± 0.7011 | 3.1656 ± 0.0457 |
| **1024** | 2.0228 ± 0.5684 | 1.3167 ± 0.1341 | $\boxed{6.3756 \pm 8.4676}$ | 3.2896 ± 0.1396 |
| **2048** | 1.7218 ± 0.0288 | 1.5416 ± 0.1405 | 1.9099 ± 0.0898 | 3.4844 ± 0.1113 |
| **4096** | 1.1411 ± 0.0404 | 1.4010 ± 0.0788 | 2.0062 ± 0.1562 | 3.8562 ± 0.3197 |
| **8192** | 1.1105 ± 0.0078 | 1.4818 ± 0.0759 | 2.3671 ± 0.1897 | 3.8474 ± 0.4784 |
| **16384** | 1.1183 ± 0.0280 | 1.7308 ± 0.1043 | 2.5833 ± 0.1737 | 4.9724 ± 0.8955 |
| **32768** | 1.1162 ± 0.0084 | 2.2099 ± 0.1880 | 3.2062 ± 0.2029 | 5.3719 ± 0.2875 |
| **65536** | 1.7463 ± 1.2217 | $\boxed{4.7474 \pm 3.1960}$ | 4.3198 ± 0.2926 | 6.8136 ± 0.2499 |
| **131072** | 1.1027 ± 0.0141 | 2.6375 ± 0.1531 | 5.7004 ± 0.2681 | 9.6912 ± 1.4337 |
| **262144** | 1.1006 ± 0.0118 | 3.3172 ± 0.1164 | 7.4630 ± 0.2309 | 10.2781 ± 0.2278 |

## JNI

- Overhead not significant
- JNI implementation differ between VMs
- Previous studies have reporter larger execution times
- Resolution of Java Timer

## Libraries (Java)

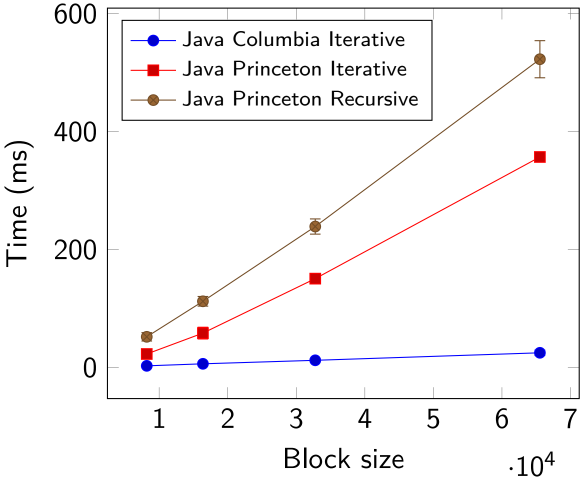


Table: Java line graph for *large* block sizes with standard deviation error bars

# Libraries (C++)

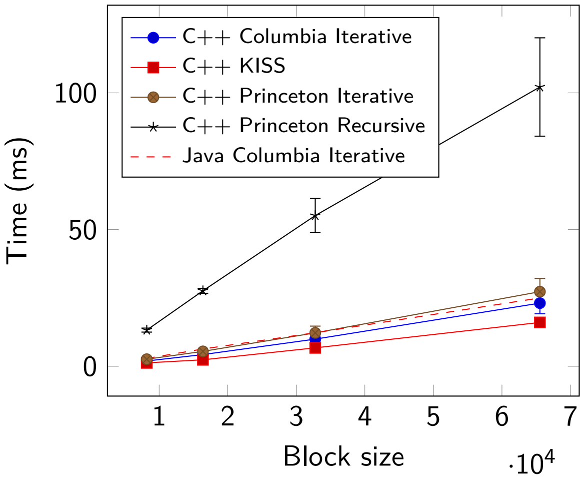


Figure: C++ line graph for *large* block sizes with standard deviation error bars

## Libraries

- Recursive algorithm worst
- Princeton algorithms allocates during run loop
- Java Columbia Iterative comparable with C++
- Translating Java $\rightarrow$ C++

## NEON

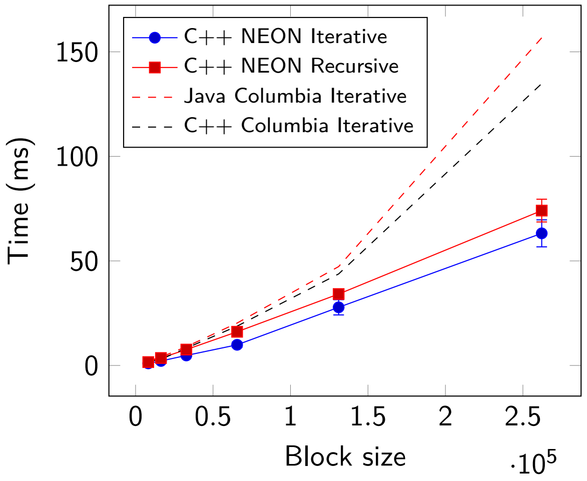


Figure: NEON results table for *extra large* block sizes, Time (ms)

## NEON

- Vectorization is clearly faster
- Columbia Iterative diverges for larger execution times
- Vectorization tests require some setup

## Conclusions

### Conclusion 1

*The overhead from JNI does not have a significant effect on performance.*

### Conclusion 2

*Of the tested algorithms, choose Columbia Iterative.*

### Conclusion 3

*Avoid allocating memory in the run-loop of a recurring task.*

### Conclusion 4

*NEON optimization is significantly faster than non-optimized code for larger block sizes.*

Questions?