WiBench: An Open Source Kernel Suite for Benchmarking Wireless Systems

Qi Zheng*, Yajing Chen*, Ronald Dreslinski*, Chaitali Chakrabarti†, Achilleas Anastasopoulos*, Scott Mahlke*, Trevor Mudge*

*University of Michigan, Ann Arbor
†Arizona State University, Tempe

IISWC ‘13
Sep 24, 2013
Mobile Device Applications

Google
facebook
twitter
yelp
amazon.com
CNN
ESPN
Mobile Subscription Growth

Different Protocols

- GSM
- 3G
- 4G (100mbps-1GBps)
- Wi-Fi
- Bluetooth
- ZigBee

GLOBAL SYSTEM FOR MOBILE COMMUNICATIONS

University of Michigan
Wireless System
Wireless Benchmark

- Requirement
  - Expose computation characteristics
  - Include a variety of kernels
  - Support different system configurations
  - Easy to use for computer architects
Drawbacks of Existing Works

✗ Incomplete list of kernels

✗ Out-dated kernels

✗ Lack of kernel details

✗ Lack of a complete system

✗ Not free
Drawbacks of Existing Works

- Incomplete list of kernels
- Out-dated kernels
- Lack of kernel details
- Lack of a complete system
- Not free

Our solution -- WiBench
WiBench (“Y-Bench”)

- Open Source Kernel Suite for Wireless Systems
- Covers the key signal processing kernels
  - 802.11a/WCDMA/LTE
- Support different configurations in each kernel
  - BPSK/QPSK/16QAM/64QAM
  - 40 – 6144 Turbo codeword length
  - $2^a \cdot 3^b \cdot 5^c \cdot 7^d$ FFT/IFFT size
- Implement the physical layer of an LTE uplink system
  - 1.56 – 100Mbps peak data rate
- Channel model
  - Communication engineers evaluate the bit error rate performance
Benefits of WiBench

✔ A variety of state-of-art kernels
✔ A complete LTE uplink system
✔ Configurability
✔ Enables hardware feature extraction
✔ Ease of use
✔ Open source and free
Outline

- Motivation
- Introduction of WiBench
  - Overview
  - Kernels
  - Application
  - Channel Model
- Demonstration of WiBench Usage
- Conclusion
## Overview

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kernels</strong></td>
<td>Channel coding/decoding</td>
</tr>
<tr>
<td></td>
<td>Rate matching</td>
</tr>
<tr>
<td></td>
<td>Scrambling/Descrambling</td>
</tr>
<tr>
<td></td>
<td>Constellation mapping/demapping</td>
</tr>
<tr>
<td></td>
<td>MIMO detection</td>
</tr>
<tr>
<td></td>
<td>FFT/IFFT</td>
</tr>
<tr>
<td></td>
<td>Sub-carrier mapping/demapping</td>
</tr>
<tr>
<td></td>
<td>Channel Estimation</td>
</tr>
<tr>
<td><strong>Channel models</strong></td>
<td>Gaussian Random Channel model (GRC)</td>
</tr>
<tr>
<td></td>
<td>Extended Pedestrian A model (EPA)</td>
</tr>
<tr>
<td></td>
<td>Extended Vehicular A model (EVA)</td>
</tr>
<tr>
<td></td>
<td>Extended Typical Urban model (ETU)</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>LTE uplink</td>
</tr>
</tbody>
</table>
Kernel Selection

802.11a

WCDMA

LTE
Kernel Selection

802.11a

WCDMA

LTE
Kernel – Channel Coding

- Control errors in data transmission

- Turbo code
  - Decoding algorithm
    - MAX-LOG-MAP algorithm
    - Iteratively compute the logarithmic likelihood of each bit

- Configuration
  - 1/3 code rate
  - Codeword length: 40 – 6144
Kernel – Scrambling

- Encrypt and randomize data
- Element-wise product
Kernel – Constellation Mapping

- Make the signals match the channel characteristics

- Mapping/Demapping
  - Binary bits $\leftrightarrow$ Complex value

- Configuration
  - BPSK
  - QPSK
  - 16QAM
  - 64QAM
Kernel – MIMO

- Improve data transmission rate and quality

- Detection algorithm
  - Least square based detection
  - Tree-based sphere detection

- Configuration
  - Support 1x1, 2x2 and 4x4 configurations
Kernel – FFT

- Fast algorithm for DFT

- Use FFTW library
  - C library to maximize performance

- Configuration
  - \(2^a \cdot 3^b \cdot 5^c \cdot 7^d\) FFT/IFFT size
Kernel – Channel Estimation

- Estimate the channel state information

- Pilot-based channel estimation
  - Least Square
    - Less complexity
  - Minimum Mean Square Error
    - More accurate estimation
Application – LTE Uplink System

- Demonstration platform
  - System configurations support peak data rates from 1.56 to 100 Mbps
Channel Models

- Standard channel models
  - Gaussian Random Channel model
  - Extended Pedestrian A model
  - Extended Vehicular A model
  - Extended Typical Urban model

- Users can evaluate the performance of different systems for different channel models
  - Bit error rate vs. Signal-to-Noise ratio
Outline

- Motivation
- Introduction of WiBench
  - Overview
  - Kernels
  - Application
  - Channel Model
- Demonstration of WiBench Usage
- Conclusion
# Experimental Configurations

<table>
<thead>
<tr>
<th>Feature</th>
<th>Desktop platform (Intel i7)</th>
<th>Mobile platform (Intel Atom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>3.40 GHz</td>
<td>1.60 GHz</td>
</tr>
<tr>
<td>Out-of-order</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Single core issue width</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I cache</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>D cache</td>
<td>32 KB</td>
<td>24 KB</td>
</tr>
<tr>
<td>L2 cache</td>
<td>256 KB</td>
<td>512 KB</td>
</tr>
<tr>
<td>Main memory</td>
<td>16 GB DDR3</td>
<td>4 GB SDRAM</td>
</tr>
</tbody>
</table>
The configuration is for 100Mbps peak data rate
Vectorization Impact (i7)

- Automatic vectorization by compiler

![Diagram showing vectorization speed up on i7 (over non-vectorization) for various tasks such as Constellation Demapping, Descrambling, Channel estimation, LS detection, Tree-based detection, Rate matching, FFT, IFFT, Turbo decoder.]
Vectorization Impact (i7)

- Automatic vectorization by compiler
Vectorization Impact (i7)

- Automatic vectorization by compiler
Architectural Implications

- A specific hardware accelerator for Turbo decoder
- Efficiently execute constellation demapping and equalization
- Include a wide SIMD engine to take advantage of DLP in the signal processing kernels
The configuration is for 100Mbps peak data rate
Different Configurations

- Keep the peak data rate same
- Change the system configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>FFT</th>
<th>IFFT</th>
<th>MIMO</th>
<th>Constellation Demapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>256</td>
<td>150</td>
<td>2X2</td>
<td>16QAM</td>
</tr>
<tr>
<td>B</td>
<td>512</td>
<td>300</td>
<td>1X1</td>
<td>16QAM</td>
</tr>
<tr>
<td>C</td>
<td>512</td>
<td>300</td>
<td>2X2</td>
<td>QPSK</td>
</tr>
<tr>
<td>D</td>
<td>1024</td>
<td>600</td>
<td>1X1</td>
<td>QPSK</td>
</tr>
</tbody>
</table>
Different Configurations (i7)

Kernel runtimes of processing an LTE uplink subframe at 12.5 Mbps on i7 in ms (except Turbo decoder)

- A: 256_2x2_16QAM
- B: 512_1x1_16QAM
- C: 512_2x2_QPSK
- D: 1024_1x1_QPSK

- Blue: Constellation demapping
- Red: Equalization
- Yellow: Predecoding
- Green: SC-FDMA
- Light green: Descrambling
- Orange: Sub-carrier demapping
- Pink: Rate matching
Different Configurations (i7)

Kernel runtimes of processing an LTE uplink subframe at 12.5 Mbps on i7 in ms (except Turbo decoder)

- **A:256_2x2_16QAM**
- **B:512_1x1_16QAM**
- **C:512_2x2_QPSK**
- **D:1024_1x1_QPSK**

- **Blue** = Constellation demapping
- **Red** = Equalization
- **Yellow** = Predecoding
- **Green** = SC-FDMA
- **Green** = Descrambling
- **Orange** = Sub-carrier demapping
- **Brown** = Rate matching
Different Configurations (i7)

Kernel runtimes of processing an LTE uplink subframe at 12.5 Mbps on i7 in ms (except Turbo decoder)

- A: 256_2x2_16QAM
- B: 512_1x1_16QAM
- C: 512_2x2_QPSK
- D: 1024_1x1_QPSK

Legend:
- Blue: Constellation demapping
- Red: Equalization
- Yellow: Predecoding
- Green: SC-FDMA
- Greenish: Descrambling
- Orange: Sub-carrier demapping
- Pink: Rate matching

University of Michigan
Outline

- Motivation
- Introduction of WiBench
  - Overview
  - Kernels
  - Application
  - Channel Model
- Demonstration of WiBench Usage
- Conclusion
We designed an open source configurable wireless signal processing kernel suite

We included an LTE uplink in the benchmark that illustrates how to build a wireless application by assembling kernels

WiBench provides several standard channel models

We provided a demonstration of WiBench usage for hardware design
Thanks!

Any questions?

http://wibench.eecs.umich.edu
Backup
Vectorization Impact (Atom)

Vectorization Speed Up on Atom (over non-vectorization):

- Constellation Demapping
- Descrambling
- Channel estimation
- LS detection
- Tree-based detection
- Rate matching
- FFT
- IFFT
- Turbo decoder
LTE uplink runtime breakdown (Atom)

- Turbo decoder: 72.64%
- Constellation demapping: 13.06%
- Equalization: 9.91%
- Other: 4.39%

- SC-FDMA: 1.82%
- Predecoding: 1.42%
  - Descrambling: 0.23%
  - Rate matching: 0.18%
  - Subcarrier demapping: 0.64%
Different configurations (Atom)

Kernel runtimes of processing one LTE uplink subframe at 12.5 Mbps on Atom in ms (except Turbo decoder)

- A: 256_2x2_16QAM
- B: 512_1x1_16QAM
- C: 512_2x2_QPSK
- D: 1024_1x1_QPSK

Legend:
- Blue: Constellation demapping
- Red: Equalization
- Yellow: Predecoding
- Green: SC-FDMA
- Light green: Descrambling
- Orange: Sub-carrier demapping
- Pink: Rate matching
Related Work

- **MiBench**
  - Only contains a small portion of the key kernels in a wireless signal processing system

- **LTE Uplink Receiver PHY Benchmark**
  - Aim to simulate the workload change in an LTE base station
  - Miss the details of several important kernels
  - Turbo decoder is represented simply as a sleep function

- **BDTI OFDM receiver benchmark**
  - Not free
Related Work

- GNU Radio
  - A free and open source software toolkit providing signal processing blocks for software radio implementation.

- Different goals
  - GNU Radio – designed to implement software defined radio on commodity hardware.
  - WiBench -- Guide hardware design of domain specific solutions.

- Building blocks
  - GNU Radio – currently not provide kernels
  - WiBench – well-built key kernels of mainstream wireless protocols as well as a complete LTE uplink system

- WiBench is more efficient sometimes
  - Turbo decoder
  - WiBench run over 20% less instructions than GNU Radio