Massively Parallel Signal Processing for Wireless Communication Systems

Michael Wu, Guohui Wang, Joseph R. Cavallaro
Department of ECE, Rice University
Wireless Communication Systems
Wireless Communication Systems

- Used in many standards:
  - WiFi 802.11ac
  - LTE
  - WiMAX

- Heavy computations on RX side:
  - MIMO Detector: decouple streams to provide estimates of the tx bits.
  - Channel Decoder: correct errors using redundant data.
Related Research Work

- MIMO detector: [SAMOS 2010 IEEE-TVT 2012]
- Turbo decoder: [CASES 2010, VLSID 2012]
- SDR systems: [IEEE Comm. Mag 2010, ISRSP 2011]

Our previous work

- Turbo decoder: [SiPS 2010, JSPS 2011]
- LDPC decoder: [SASP 2011, Asilomar 2011]
- NB-LDPC decoder: [Asilomar 2012]
Massively parallel implementations

- Massively parallel implementations:
  - Tailored algorithms to improve efficiency.
  - Results:
    - Achieve high throughput (faster than existing work)
    - Very flexible, can be a good platform for SDR systems.
Outline

- MIMO soft detection algorithm on GPU
  - Introduction to MIMO detection
  - Kernel mapping
  - Optimization techniques
  - Experiment results

- Multi-standard LDPC Decoder on GPU
  - Introduction to LDPC algorithm
  - Kernel mapping
  - Optimization techniques
  - Experiment results
Modulation

- Encode data in amplitude and phase of a sinusoid

- Higher modulation order → more data per symbol
MIMO System Model

- **Spatial Multiplexing:** ↑ throughput by transmitting multiple streams
- **Receiver:** Transmit streams interfere with each other

\[
\begin{bmatrix}
 y_0 \\
 y_1 \\
 y_2 \\
 y
\end{bmatrix} = \begin{bmatrix}
 h_{00} & h_{01} & h_{02} \\
 h_{10} & h_{11} & h_{12} \\
 h_{20} & h_{21} & h_{22}
\end{bmatrix} \begin{bmatrix}
 x_0 \\
 x_1 \\
 x_2
\end{bmatrix} + \begin{bmatrix}
 n_0 \\
 n_1 \\
 n_2
\end{bmatrix}
\]
MIMO-OFDM

Break a wideband signal into many independent subcarriers
Perform MIMO detection independently many times, one per subcarrier
Many subcarriers for many wireless standards.

- LTE 20Mhz subframe: 14*1200 subcarriers
MIMO Detection

\[
\begin{bmatrix}
\hat{x}_0 \\
\hat{x}_1 \\
\hat{x}_22
\end{bmatrix} = \begin{bmatrix}
h_{00} & h_{01} & h_{02} \\
h_{10} & h_{11} & h_{12} \\
h_{20} & h_{21} & h_{22}
\end{bmatrix} \begin{bmatrix}
x_0 \\
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
m_0 \\
m_1 \\
m_2
\end{bmatrix}
\]

\[
\hat{y} = Rx + n
\]

- Probability of a path, \( x \), is **inversely** prop. to \( d \):
  \[
d = \| \hat{y} - R(x) \|^2 = \sum_i d_i
\]

- Probabilities of all paths are used to generate bit probability values.

\[
4 \times 4 \ 64\text{QAM} \rightarrow 64^4 = 16,777,216 \text{ paths}
\]
Selective Spanning with Fast Enumeration (SSFE)*

- Real value decomposition
- Data parallel deterministic search

Generate M likely paths which are used to generate bit probability values

1\textsuperscript{st} level: enumerate all modulation points.

Subsequent levels: depth-first search, pick the best outgoing node
SSFE Detector: Node Expansion

- Find best node—$x_0$—that minimizes the cumulative distance
  - Pick the constellation point closest to the zero forcing solution.

Zero forcing solution: 
$$
\hat{x}_0 = \frac{1}{r_{00}} (\hat{y}_0 - r_{02}x_2 - r_{01}x_1)
$$

64 QAM example: 
$$
\hat{x}_0 = 3.9
$$

-7 -5 -3 -1 1 3 5 7

Schnoor-Euchner enumeration
GPU Implementation

- Search algorithm maps well onto GPU
  - Data parallel with no sharing
    - Each search path is independent
  - Efficient node expansion
    - Complexity doesn’t depend on modulation
  - Modest storage requirement

M threads per detection
GPU Implementation of SSFE

```c
// enumerate a modulation point for 1st antenna
path[0] = mod_table[tid%8];
path[1] = mod_table[tid/8];
dist += calc_dist(y, r, path[0]);
dist += calc_dist(y, r, path[1]);

// depth first search
For i = 2:ntx
    // compute partial Euclidean dist
    ped = 0;
    For j = 0:i
        ped += calc_dist(y, r, path[j]);
    
    // find best outgoing path
    Path[j] = SE_expand(dist, r);
    dist = update_dist(dist, ped);
```

One thread block handles one subcarrier

- Spawns 1 thread per modulation point (M threads)
- Completely unrolled inner and outer loops
- Path stored in registers

Demodulator + soft estimate computation not shown

Result

- 4x4 16QAM: 940Mbps
- 4x4 64QAM: 480Mbps
N-Way MIMO Detector

- Duplicate search block depending on FER requirement.
  - Add permute block which enforces a detection order
  - Example: N = 2, two search blocks
  - Larger lists, NM candidates

Qi, Q., Chakrabarti, C, *Parallel high throughput soft-output sphere decoder*, (SIPS 10)
M Wu, C Dick, JR Cavallaro, *Improving MIMO sphere detection through antenna detection order scheduling* (SDR Forum 11)
GPU Implementation of $N$-Way MIMO Detector

- Duplicate threads to improve accuracy of the detection algorithm

- Divide a thread block into $N$ subsections

- Each subsection consists of $M$ threads operates on a different channel permutation
  - Performs SSFE detection independently
  - Generate a $NM$ size candidate list.
N-Way MIMO FER Performance

- Soft Output detectors + Rate 1/2 WiMAX LDPC code, Rayleigh fading channel.
  - 1 outer iteration + 20 inner iteration with early termination
  - Compared to soft-output K-best and exhaustive (MAP) detector.

- **4x4 16QAM**

- **4x4 64QAM**
N-Way MIMO Detector Throughput

- GK104, 1536 SM @ 915MHz, 256-bit DDR5 @ 6Gbps
- 8192 subcarriers, Kernel time only
N-Way MIMO Detector Throughput vs Workload

- GK104, 1536 SM @ 915MHz, 256-bit DDR5 @ 6Gbps
- Kernel time only
Performance Comparison

<table>
<thead>
<tr>
<th>Number of Subcarriers</th>
<th>16QAM (Mbps)</th>
<th>64QAM (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FPFSD*</td>
<td>Ours N=4</td>
</tr>
<tr>
<td>150*7</td>
<td>72.41</td>
<td>232.27</td>
</tr>
<tr>
<td>300*7</td>
<td>82.56</td>
<td>246.60</td>
</tr>
<tr>
<td>600*7</td>
<td>90.44</td>
<td>256.83</td>
</tr>
<tr>
<td>900*7</td>
<td>91.39</td>
<td>256.22</td>
</tr>
<tr>
<td>1200*7</td>
<td>92.31</td>
<td>256.88</td>
</tr>
</tbody>
</table>

- **FPFSD**
  - N=4—4 parallel detectors with different permutations
  - Differences: a) operates in complex domain b) one kernel for search + one kernel for soft-output generation

- **Fermi, 448 core @ 1150MHz, 320bit DDR5 @ 3Gbps**

*Sandra Roger, et.al Fully Parallel GPU Implementation of a Fixed-Complexity Soft-Output MIMO Detector (IEEE TVT 2012)*
**N-Way MIMO Detector**

- Duplicate search block depending on FER requirement.
  - Add permute block which enforces a detection order
  - Example: N = 2, two search blocks
  - Larger lists, NM candidates

Qi, Q., Chakrabarti, C, *Parallel high throughput soft-output sphere decoder*, (SIPS 10)
M Wu, C Dick, JR Cavallaro, *Improving MIMO sphere detection through antenna detection order scheduling* (SDR Forum 11)
**N-Way QR Decomposition**

Divide a thread block into N subsections

Each subsection of N threads operates on a different channel permutation

- Performs modified Gram Schmidt QR on an extended matrix $[H|Y]$
- Generate $R$ and $\hat{Y}$

QR decomposition time for 8192 symbols

<table>
<thead>
<tr>
<th></th>
<th>N=1</th>
<th>N=2</th>
<th>N=3</th>
<th>N=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>0.201 ms</td>
<td>0.350 ms</td>
<td>0.551 ms</td>
<td>0.675 ms</td>
</tr>
</tbody>
</table>

- GK104, 1536 SM @ 915MHz, 256-bit DDR5 @ 6Gbps
- Kernel time only

290.3 Mbps @ 64QAM
Complete Design

- Complete design, QR + MIMO Detection
- Also includes PCIe transfer time

- GK104, 1536 SM @ 915MHz, 256-bit DDR5 @ 6Gbps
- 8192 subcarriers
Complete Design

- Transfer time doesn’t depend on N (# of parallel search) or M (modulation size)
  - Transfer time can be hidden
- QR depends only on N
- Detection depends on N and M
Outline

- Multi-standard LDPC Decoder on GPU
  - Introduction to LDPC algorithm
  - Kernel mapping
  - Optimization techniques
  - Experiment results
Linear block codes

Encoding: \( x \cdot G = c \)
- K-bit \( x \) encoded into N-bit codeword \( c \) (K<N)
- Generator matrix \( G \)
- Parity check matrix \( H \):
  \[ H \cdot c^T = 0 \ (G \cdot H^T = 0) \]
Low-density parity-check (LDPC) codes

- Error-correction codes
- Provides near-capacity error-correcting performance

- Application of LDPC codes
  - Wireless communication
    - IEEE 802.16m WiMax
    - IEEE 802.11n, 802.11ac WiFi
  - 10Gbps Ethernet communication
    - IEEE 802.3an
  - Digital broadcast: DVB-S2
  - High speed magnetic storage device
  - Satellite communication

Challenges of decoder design

- High throughput requirement
- Multi-standard support
- Flexibility and scalability
LDPC Codes

- LDPC codes are linear block codes defined by sparse matrices $H$
- Codeword $c$ should satisfy the parity-check equations: $H \cdot c^T = 0$
- Belief propagation decoding algorithm

**Sparse matrix $H$**

$$H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1
\end{bmatrix}$$

**Tanner graph**
LDPC Decoding: Belief propagation decoding

- Bit stream
- Modulation
- Modulated symbol
- Wireless channel
- Received symbol

<table>
<thead>
<tr>
<th>Bit stream</th>
<th>0, 1, 1, 0, 1, 0, 0, 1,…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modulation</td>
<td>-1, 1, 1, -1, 1, -1, -1, 1,…</td>
</tr>
<tr>
<td>Received symbol</td>
<td>-1.3, 0.8, 1.1, -0.7, 0.5, -1.2, -0.9, 1.1,…</td>
</tr>
</tbody>
</table>

Probability($c_n=0$ | received) VS Probability($c_n=1$ | received)

- $H \cdot c^T = 0$
- Complexity $\sim O(N^3)$
Belief propagation decoding

Initialization

Check Node Processing

Variable Node Processing

Finish decoding

Done

Check Node Processing, $R_{mn}$

\[
H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Variable Node Processing, $Q_{mn}$

\[
H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Belief propagation decoding algorithm: initialization

- **Decoded Bit stream**
- **Channel Decoder**
- **Probability values**
- **Detector/Demodulator**
- **Wireless receiver**

Diagram:

- **Bit probability values**
- **CN0 to CN3**
- **VN0 to VN7**

The diagram illustrates the process of belief propagation decoding, starting with the wireless receiver and ending with the decoded bit stream.
Belief propagation decoding algorithm: CNP

\[
H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 
\end{bmatrix}
\]
Belief propagation decoding algorithm: CNP
Belief propagation decoding algorithm: VNP

\[ H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 
\end{bmatrix} \]
Belief propagation decoding algorithm: hard decision
Early Termination

- Early termination (ET)
  - Avoid unnecessary computations when codeword converges
  - Widely used in low power decoding architecture.

- ET for LDPC decoder
  - Use **parity check equation**: $H \cdot c^T = 0$
  - Use **massive threads** to perform parity check

\[ H \cdot c^T = 0 \]
Why GPU for LDPC Decoding?

**LDPC Decoding**
- Highly parallel algorithm
  - No dependency for computations among rows (or columns).
- High complexity iterative algorithm
- Clear algorithm structure

**GPU**
- SIMT Parallel architecture
- Enough workload to fully occupy the GPU’s computing resources
- Partition tasks into kernel functions.
Partition the LDPC Decoding Task

**Initialization**

- **Host code**

**Start decoding iterations**

- **Kernel 1**: Check node processing
- **Kernel 2**: Variable node processing
- **Kernel 3**: Early termination (ET) check

(Go back if the ET condition is not met)

**Make hard decision**

- **Host code**
CUDA Kernel 1: Check Node Processing

- One thread block processes one row of sub-matrices
- Each thread block contains 81 threads, each thread processes one row of the H matrix (one check node).

<table>
<thead>
<tr>
<th>( I_{57} )</th>
<th>( I_{50} )</th>
<th>( I_{11} )</th>
<th>( I_{50} )</th>
<th>( I_{79} )</th>
<th>( I_{1} )</th>
<th>( I_{0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{3} )</td>
<td>( I_{28} )</td>
<td>( I_{0} )</td>
<td>( I_{55} )</td>
<td>( I_{7} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{30} )</td>
<td>( I_{24} )</td>
<td>( I_{37} )</td>
<td>( I_{36} )</td>
<td>( I_{14} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{62} )</td>
<td>( I_{53} )</td>
<td>( I_{53} )</td>
<td>( I_{3} )</td>
<td>( I_{35} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{40} )</td>
<td>( I_{20} )</td>
<td>( I_{66} )</td>
<td>( I_{22} )</td>
<td>( I_{28} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{0} )</td>
<td>( I_{8} )</td>
<td>( I_{42} )</td>
<td>( I_{50} )</td>
<td>( I_{8} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{69} )</td>
<td>( I_{79} )</td>
<td>( I_{79} )</td>
<td>( I_{56} )</td>
<td>( I_{52} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{65} )</td>
<td>( I_{38} )</td>
<td>( I_{57} )</td>
<td>( I_{72} )</td>
<td>( I_{27} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{64} )</td>
<td>( I_{14} )</td>
<td>( I_{52} )</td>
<td>( I_{30} )</td>
<td>( I_{32} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{45} )</td>
<td>( I_{70} )</td>
<td>( I_{0} )</td>
<td>( I_{77} )</td>
<td>( I_{9} )</td>
<td>( I_{0} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{2} )</td>
<td>( I_{16} )</td>
<td>( I_{56} )</td>
<td>( I_{57} )</td>
<td>( I_{55} )</td>
<td>( I_{12} )</td>
<td>( I_{0} )</td>
</tr>
<tr>
<td>( I_{24} )</td>
<td>( I_{11} )</td>
<td>( I_{61} )</td>
<td>( I_{60} )</td>
<td>( I_{27} )</td>
<td>( I_{51} )</td>
<td>( I_{16} )</td>
</tr>
</tbody>
</table>

* 802.11n (1944, 972) LDPC code
CUDA Kernel 2: Variable Node Processing

- One thread block processes one column of sub-matrices.
- Use 1944 threads to run concurrently.

![Matrix Diagram]

- One thread block processes one column of sub-matrices.
- Use 1944 threads to run concurrently.

<table>
<thead>
<tr>
<th>Probability value memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1944 threads</td>
</tr>
</tbody>
</table>

Update variable node message
CUDA Kernel 3: Parallel Early Termination

Check equation

\[ b[0] \oplus b[1] \oplus \ldots \oplus b[M-1] = 0 \]
Decoding Algorithm Optimization

- **Loosely coupled algorithm**
  - Don’t store $q_{mn}$ in the memory.
  - Before computing $r_{mn}$, recover $q_{mn}$ first.
  - Good for CUDA implementation
    - Reduce the device memory storage
    - Reduce number of memory operations

- **Forward-backward check node update**
  - For one row with $\omega_r$ non-zero element, we need to traverse the row for $\omega_r$ times.
  - Use forward-backward algorithms
  - **Reduce number of operations:** $M \omega_r(\omega_r-2) \rightarrow M (3\omega_r-2)$
    - For example, $M=2000$, $\omega_r=7$, reduce ~50% operations.
**Optimization:** multi-codeword decoding

- Utilize the 2-D thread block structure
- Reduce diverse branches
- Take advantage of constant memory
- Good flexibility and scalability
Optimization – Efficient Storage

- Memory optimization
  - **Constant memory**: increase throughput by 8%
  - **Compact representation**

```c
struct h_element {
    byte x;
    byte y;
    byte shift_value;
    byte valid;
};
```

### H_kernel1

#### H_kernel2

- **Horizontal compression**
- **Vertical compression**
Optimization – Memory Coalescing

- Coalescing device memory access
  - Compact format of $R_{mn}$ and $\Delta_{mn}$ (check node message)
  - Writing compressed $R_{mn}$ and $\Delta_{mn}$ matrices column-wise

  → coalesced memory access (20% throughput improvement)
## Experimental Results: LDPC Decoding Throughput

<table>
<thead>
<tr>
<th>Code type</th>
<th># of iterations</th>
<th>Decoding Time (ms)</th>
<th>Decoding Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11n WiFi</td>
<td>5</td>
<td>26.0</td>
<td>74.7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>49.8</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>71.5</td>
<td>27.2</td>
</tr>
<tr>
<td>802.16m WiMAX</td>
<td>5</td>
<td>24.0</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>43.0</td>
<td>52.31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>64.0</td>
<td>36</td>
</tr>
</tbody>
</table>

* Host PC: Intel i5-750 Quad-core CPU, @2.67GHz, 8GB DDR3 memory
* GTX 470 Fermi GPU
Experiment results: Early Termination Throughput

- Adaptive ET scheme:
  - Low SNR: ET off
  - High SNR: ET on

- Increase throughput for high SNR
Throughput VS Workload

- WiMax code, 2304 bits, rate $\frac{1}{2}$ code
- At first, throughput increases almost linearly as workload increases
- After certain point, throughput stops increasing, because the threads occupies all the computation SMs in the GPU.

## Comparison with Recently Published Work

<table>
<thead>
<tr>
<th>Work</th>
<th>Code length</th>
<th>Normalized throughput (# of iterations = 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park et al, [Journal on WCN 2011]</td>
<td>18000 bits</td>
<td>2.809 Mbps</td>
</tr>
<tr>
<td>Yau et al, [ICACT 2011]</td>
<td>1/2 CMMB codes, 9126 bits</td>
<td>2.046 Mbps</td>
</tr>
<tr>
<td>Zhao et al, [ICA3PP 2011]</td>
<td>4058 bits QC-LDPC</td>
<td>1.067 Mbps</td>
</tr>
<tr>
<td>Abburi, [VLSID 2011]</td>
<td>2034 bits WiMax</td>
<td>40 Mbps</td>
</tr>
<tr>
<td>Kennedy, [journal on WCN 2012]</td>
<td>2034 bits WiMax</td>
<td>32.9 Mbps</td>
</tr>
<tr>
<td>Kang [ICC 2012]</td>
<td>2048 bits, R=0.89</td>
<td>24.09 Mbps</td>
</tr>
<tr>
<td><strong>Our work (Results on GTX 470)</strong></td>
<td><strong>2304 bits WiMax</strong></td>
<td><strong>52.31 Mbps</strong></td>
</tr>
</tbody>
</table>

Beyond Binary LDPC Codes – $GF(q)$ Nonbinary LDPC

$$H = \begin{pmatrix} 4 & 2 & 7 & 3 \\ 3 & 1 & 5 & 4 \\ 3 & 6 & 2 & 6 \\ 4 & 3 & 7 & 1 \end{pmatrix}$$

Inside one work group

Forward computation

Backward computation

Barrier local memory sync

N work groups
Conclusion

- Massively parallel implementations of a MIMO detector and a LDPC decoder on GPU
  - Tailor your algorithm
  - Tweak algorithm to improve efficiency

- Results:
  - Achieve high throughput
  - Faster than Existing work
  - Very flexible, can be a good platform for SDR systems

- Future work
  - Improving performance on Kepler
  - GPU accelerated SDR systems

- Links
  - Michael Wu: http://www.ruf.rice.edu/~mbw2/
Acknowledgement

- Research supported by US National Science Foundation under grants CNS-1265332, ECCS-1232274, EECS-0925942 and CNS-0923479.

- Equipment donations generously provided by NVIDIA.