A GPU Implementation of Belief Propagation Decoder for Polar Codes

Bharath Kumar Reddy L. and Nitin Chandrachoodan

Indian Institute of Technology Madras

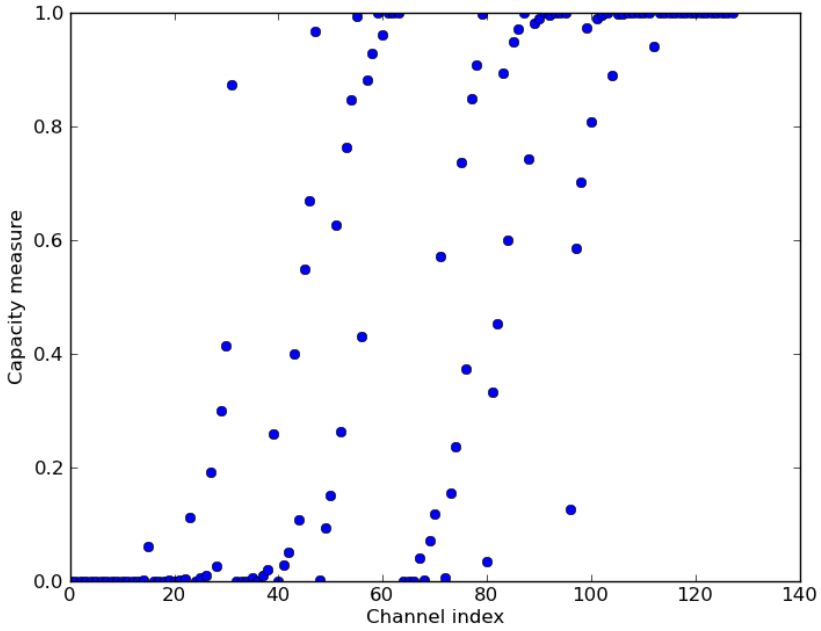
Nov 6, 2012

Outline

Topic

Polar Codes

- Capacity achieving codes for Symmetric binary-input discrete memoryless channels (B-DMC) [1]
- Capacity is achieved under Successive Cancellation(SC) decoding for very large code lengths (2^{20} or more bits)
- Objective : To implement a fast decoder for polar codes



Channel capacity polarization as a function of channel instance.

[1] E. Arıkan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels", IEEE Trans. Info. Theory, 2009

Decoding algorithms

Successive Cancellation (SC) decoder

- Serial – bit-by-bit decoding
- Complexity $O(N \log N)$
- Poor parallelism
- Good performance only for very large block lengths $> 2^{20}$

Decoding algorithms

Successive Cancellation (SC) decoder

- Serial – bit-by-bit decoding
- Complexity $O(N \log N)$
- Poor parallelism
- Good performance only for very large block lengths $> 2^{20}$

Belief Propagation (BP)

- Generic algorithm based on *message passing*
- Performs well at practical block lengths (100-1000 bits)
- Many stages can be implemented in parallel as there is no interdependence among the bits
- Iterative: may require more iterations to converge

GPUs

Graphic Processing Unit

- Many-core processors an array of multithreaded Streaming Multiprocessors (SM)
- Multiple levels of memory: registers$<$ shared memory$<$ global memory
- Synchronization among SMs is possible only via global memory
- Good for applying same computation on a large set of data

GPUs

Graphic Processing Unit

- Many-core processors an array of multithreaded Streaming Multiprocessors (SM)
- Multiple levels of memory: registers< shared memory< global memory
- Synchronization among SMs is possible only via global memory
- Good for applying same computation on a large set of data

Our Specification

- NVIDIA GTX 560 Ti - 384 cores clocking at 1.66GHz
- Fermi architecture : Max of 1536 concurrent threads, Max of 1024 threads per block, Max of 8 blocks per SM

Assumptions

Assumptions

- We have a large number of codewords available to be decoded
- Calculations are done assuming Likelihood Ratios are available as floating point numbers
- Rate 1/2 coding
- An encoder structure based on recursive definition

Encoding Graph

$c = uG$, where G, generator matrix, $= F^{\otimes n}$, n^{th} Kronecker power of $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$

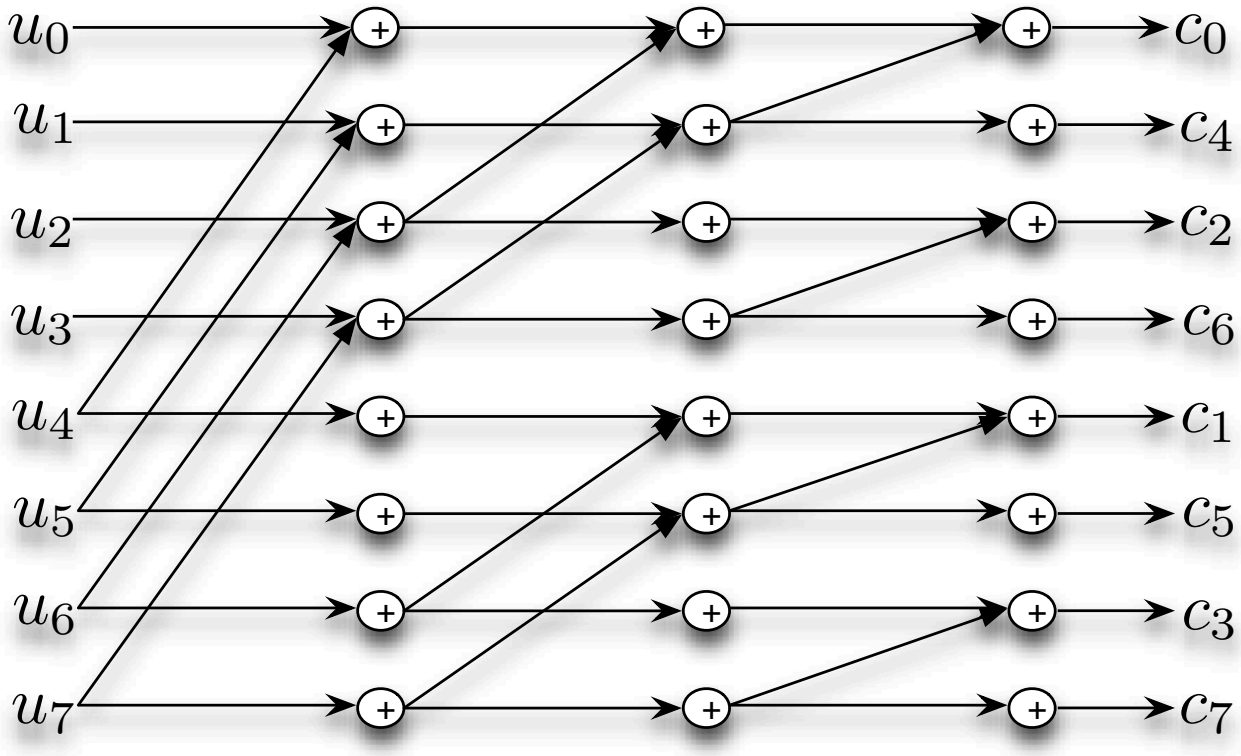


Figure : Polar Code Encoder for length 8

Encoder

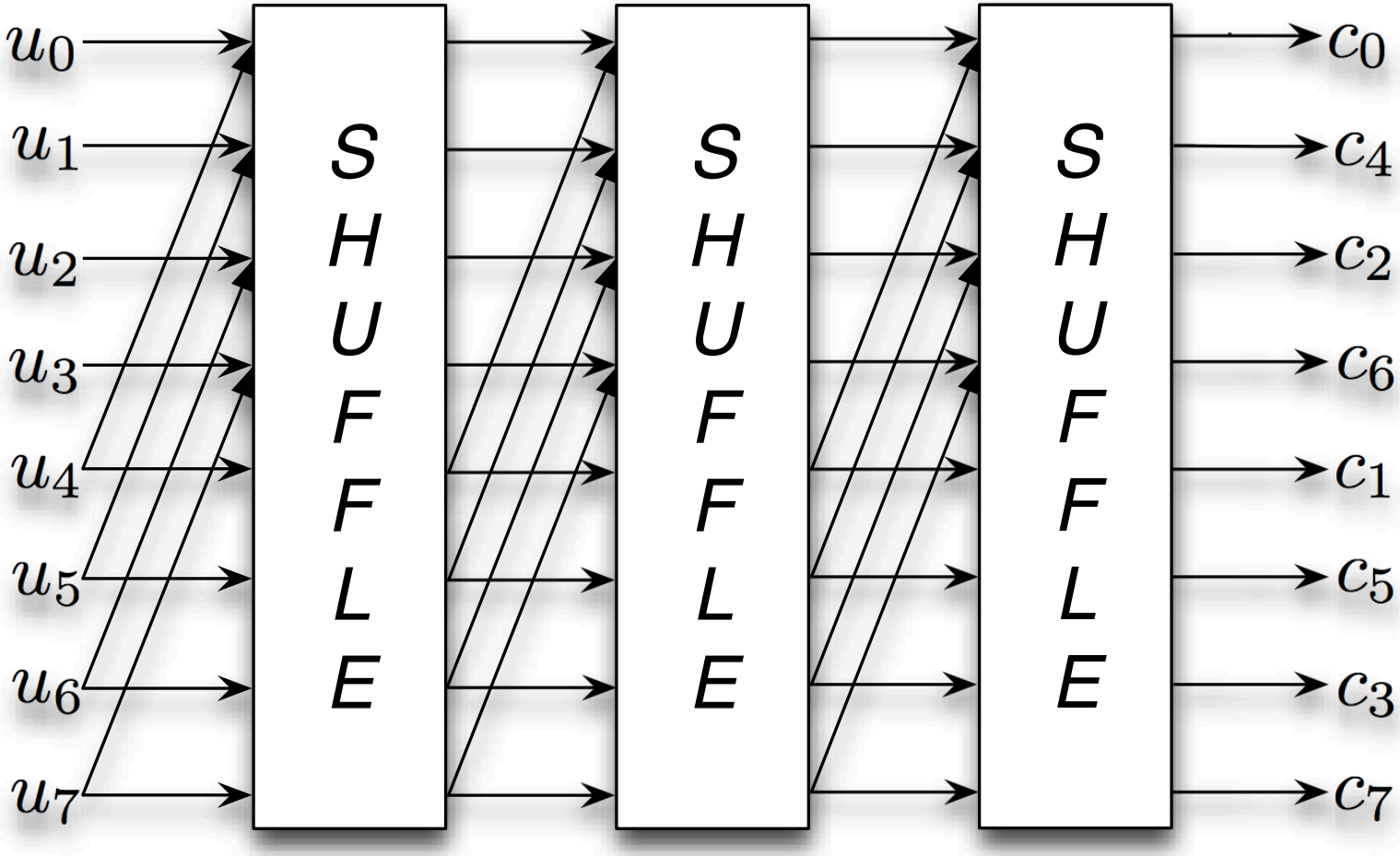
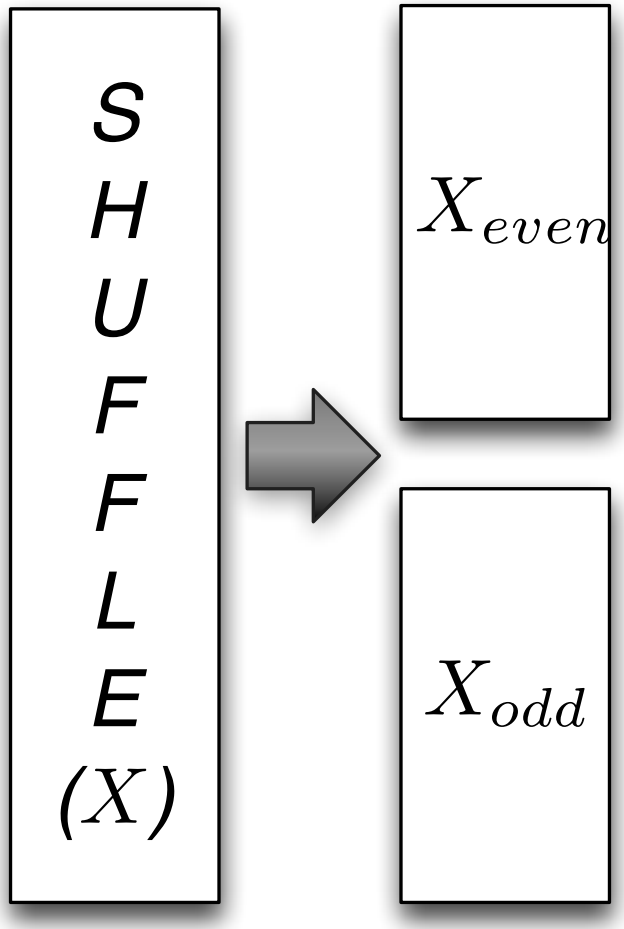
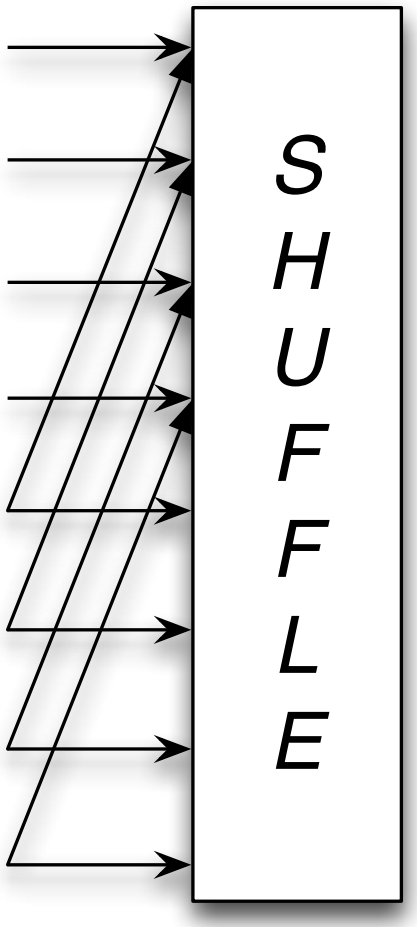


Figure : An alternate way of representing the encoder

Encoder

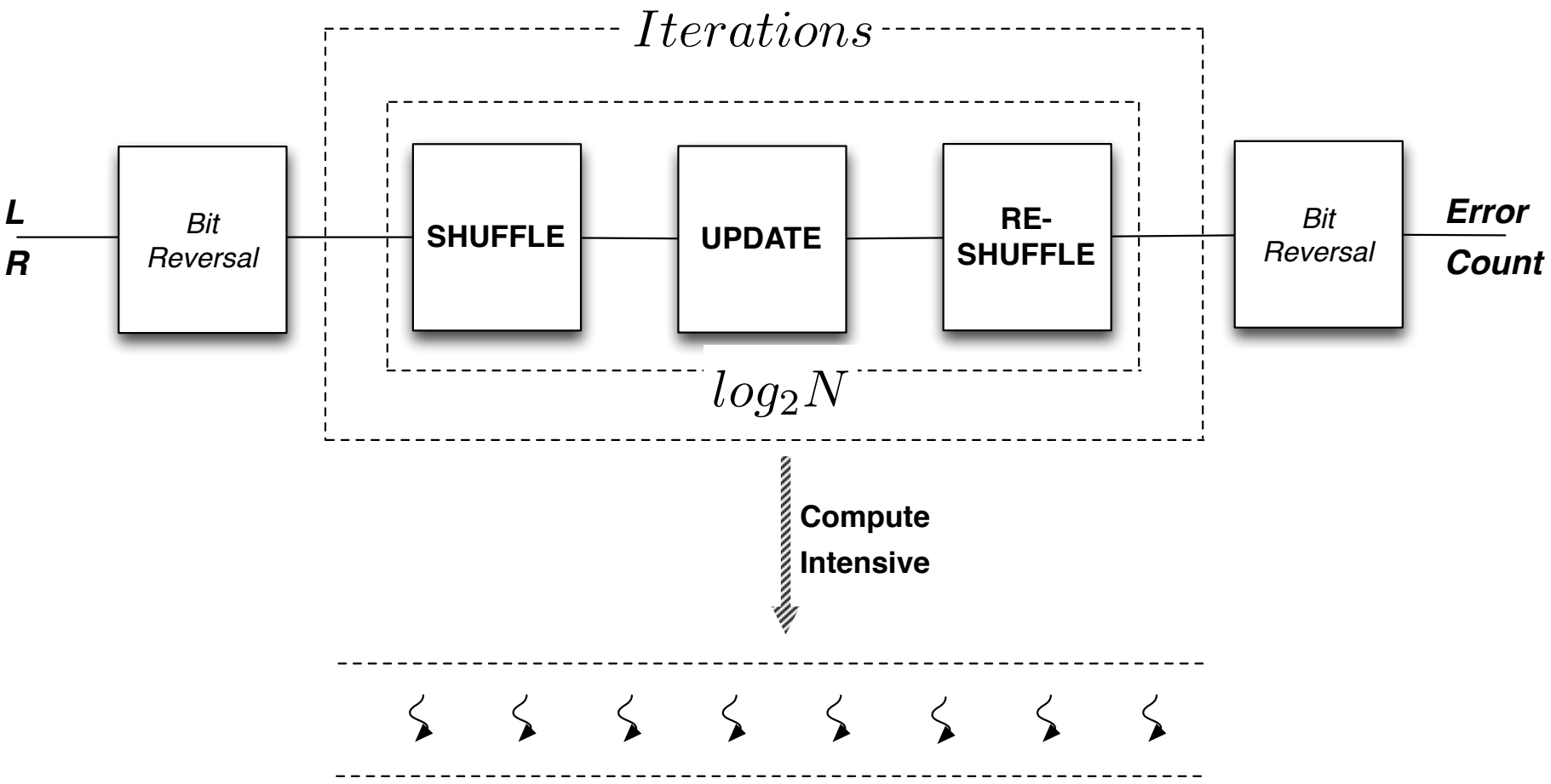


Unit of repitition.
This is repeated $log_2 N$ for each iteration



Topic

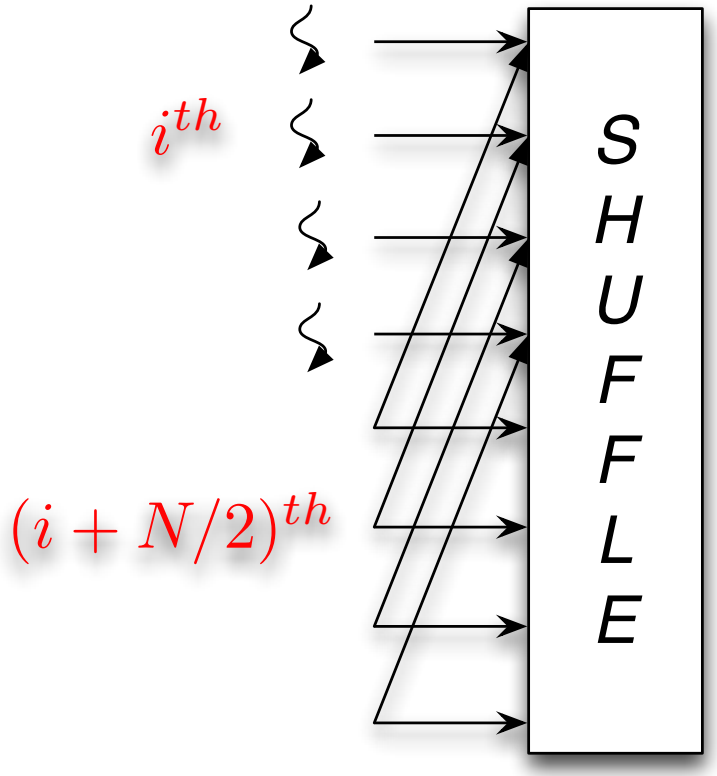
Overview


Iterations
L
R
Bit Reversal
SHUFFLE
UPDATE
RE-SHUFFLE
Bit Reversal
Error Count
$log_2 N$
Compute Intensive

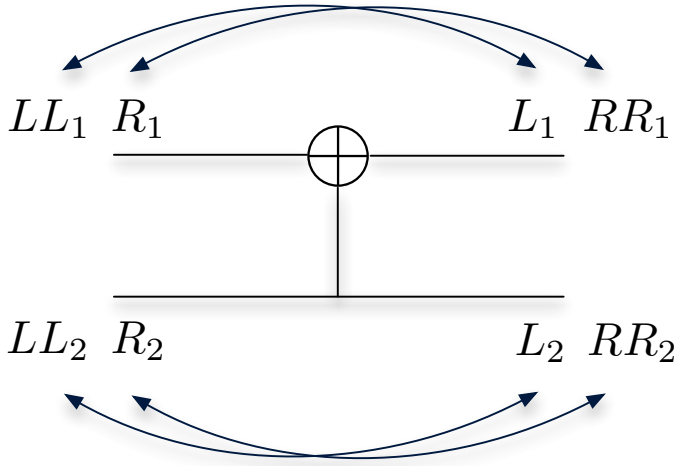
Identifying parallelism

Thread Level Parallelism

- Decoding a codeword using inherent parallelism
- i^{th} thread updates i^{th} and $(i + N/2)^{th}$ nodes
- To decode a N-length codeword, N/2 threads are utilized



Belief update

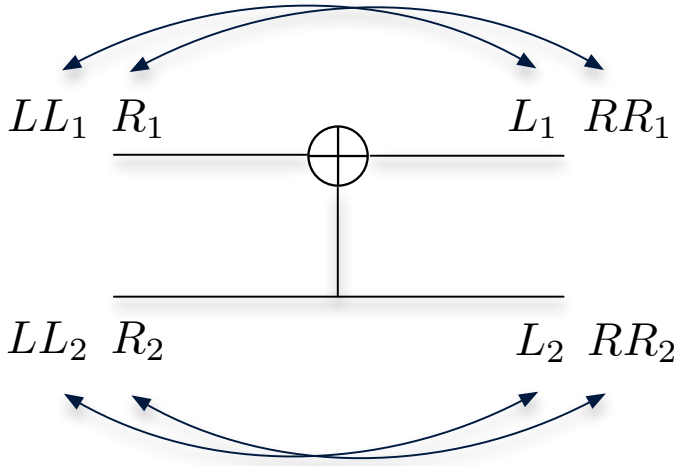


Messages

- Likelihood ratios as basis for messages
- R_i left-to-right (frozen bits)
- L_i right-to-left (from channel)
- Sum-product equations

$$
\begin{aligned}
RR_1 &= \frac{1 + R_1 R_2 L_2}{R_1 + R_2 L_2} \\
RR_2 &= R_2 . \frac{1 + R_1 L_1}{R_1 + L_1} \\
LL_1 &= \frac{1 + L_1 L_2 R_2}{L_1 + L_2 R_2} \\
LL_2 &= L_2 . \frac{1 + L_1 R_1}{L_1 + R_1}
\end{aligned}
$$

Belief update



$$
\begin{aligned}
RR_1 &= \frac{1 + R_1 R_2 L_2}{R_1 + R_2 L_2} \\
RR_2 &= R_2 . \frac{1 + R_1 L_1}{R_1 + L_1} \\
LL_1 &= \frac{1 + L_1 L_2 R_2}{L_1 + L_2 R_2} \\
LL_2 &= L_2 . \frac{1 + L_1 R_1}{L_1 + R_1}
\end{aligned}
$$

Messages

- Likelihood ratios as basis for messages
- R_i left-to-right (frozen bits)
- L_i right-to-left (from channel)
- Sum-product equations

LR or *LLR*?

- Avoid Jacobean computation (or approximation)
- Floating point multiplication not expensive on GPU
- *LLR* less susceptible to dynamic range problems

Memory management

Shared Memory

- On-chip memory
- Very low access latency compared to global memory
- Limited - *48KB per SM*
- All computations in the shared memory
- Bank conflicts are avoided

Memory management

Shared Memory

- On-chip memory
- Very low access latency compared to global memory
- Limited - *48KB per SM*
- All computations in the shared memory
- Bank conflicts are avoided

Table : Speed up using shared memory against global memory (time in *ms*)

Length	Global memory	Shared memory	Speed-up
256	74.17	7.41	10
512	101.37	8.94	11
1024	234.66	20.5	12
2048	825.96	60.98	14

Identifying parallelism

Block Level Parallelism

- Decoding as many codewords as that could fit in shared memory

Table : #blocks launched with varying code lengths

Length (N)	Shared mem/ codeword	# blocks (≤ 8)	# simultaneous codewords
256	2KB	$\frac{1536}{128} = 8$ $(12 > 8)$	24
512	4KB	$\frac{1536}{256} = 6$	12
1024	8KB	$\frac{1536}{512} = 3$	6
2048	16KB	$\frac{1536}{1024} = 1$	3

Memory Management

Registers

- Fastest form of storage on GPU
- Limited (32K) per SM
- More registers per thread - less number of concurrent threads
- For the Fermi architecture, if a thread uses 20 or less registers, then all threads are active

Memory Management

Registers

- Fastest form of storage on GPU
- Limited (32K) per SM
- More registers per thread - less number of concurrent threads
- For the Fermi architecture, if a thread uses 20 or less registers, then all threads are active

Table : Number of registers used

Length	# reg/thread	# active threads
256	22	1408 (91.66%)
512	22	1280 (83.33%)
1024	22	1024 (66.67%)
2048	22	1024 (66.67%)

Memory Management

Fast math operations, Intrinsics and Instruction Optimizations

- Functions replaced by their intrinsics
- Registers used per thread - 22
- Registers used per thread after these optimizations - 19

Memory Management

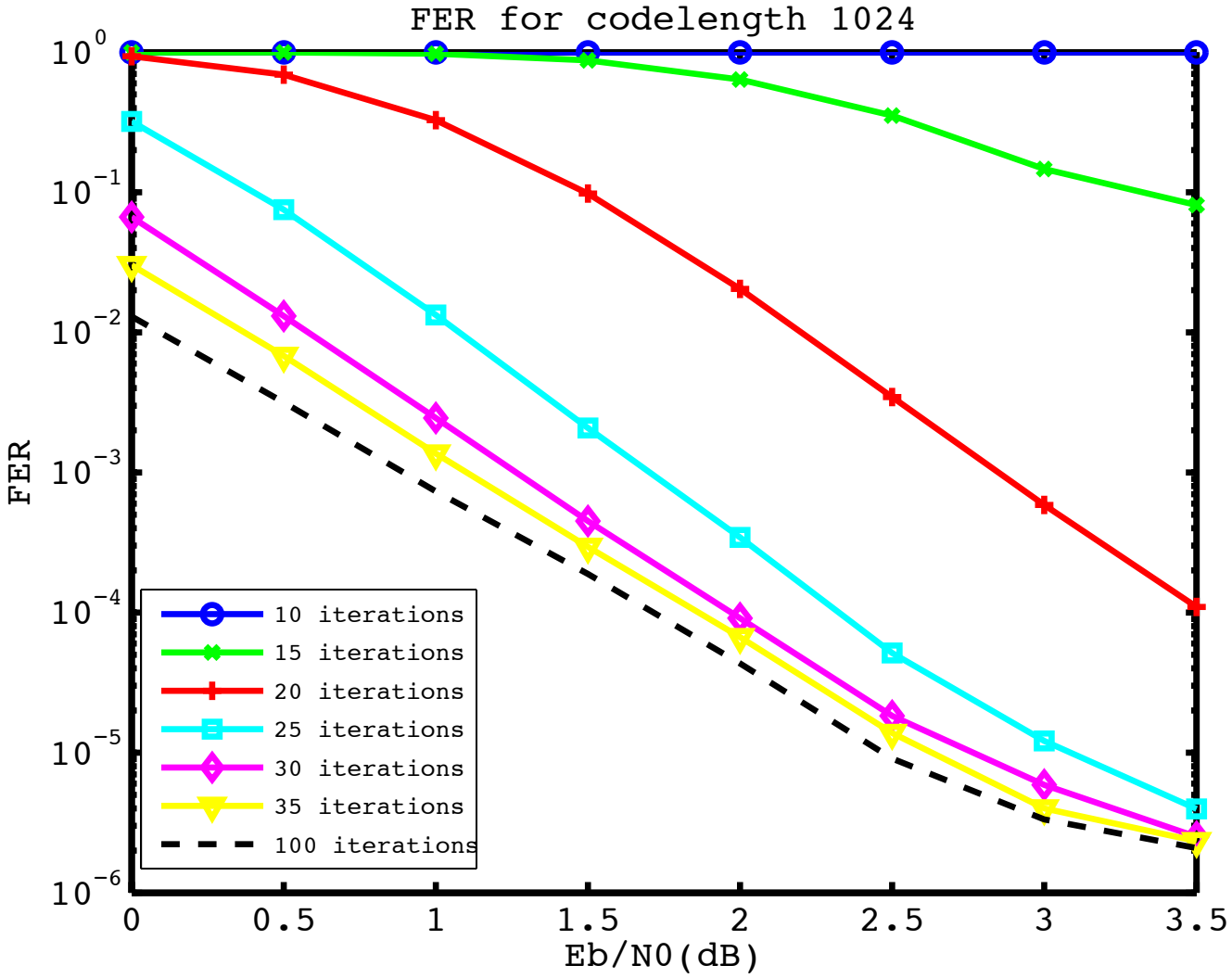
Fast math operations, Intrinsics and Instruction Optimizations

- Functions replaced by their intrinsics
- Registers used per thread - 22
- Registers used per thread after these optimizations - 19

Table : Speed-up using optimizations (for 35 iterations)

Length	Throughput	Speedup
256	17.57	1.1
512	8.71	1.2
1024	3.55	1.5
2048	1.23	-

FER vs iterations



Results

Optimizations done

- Right choice of decoder architecture for thread level parallelism
- Shared memory usage tuned for block level parallelism
- Reducing register count using approximate fast math operations

Results

Optimizations done

- Right choice of decoder architecture for thread level parallelism
- Shared memory usage tuned for block level parallelism
- Reducing register count using approximate fast math operations

Table : Throughput (Mbps) Performance with iterations

Length	10	15	20	25	30	35
256	57.20	38.82	30.34	24.42	20.32	17.57
512	29.08	19.98	15.01	12.08	10.15	8.71
1024	11.85	8.06	6.04	4.923	4.13	3.55
2048	4.089	2.79	2.11	1.71	1.43	1.23

Topic

Summary

- We have described a parallel implementation of a decoder for Polar Codes using GPU
- Using the right kind of architecture, a single stage can be reused
- We have also applied optimizations to the usage of registers and shared memory to get a good throughput
- The resulting decoder is much faster than a CPU decoder and scales with cores available provided enough codewords are available for decoding

Future Work

- Working with larger block lengths
- Codewords spills out of shared memory
- Comparing with LLRs
- Optimizing the BP update equations

Acknowledgment

We would like to thank Dr. Andrew Thangaraj, IIT Madras, for his valuable suggestions and insights during the course of this work

Thank you!

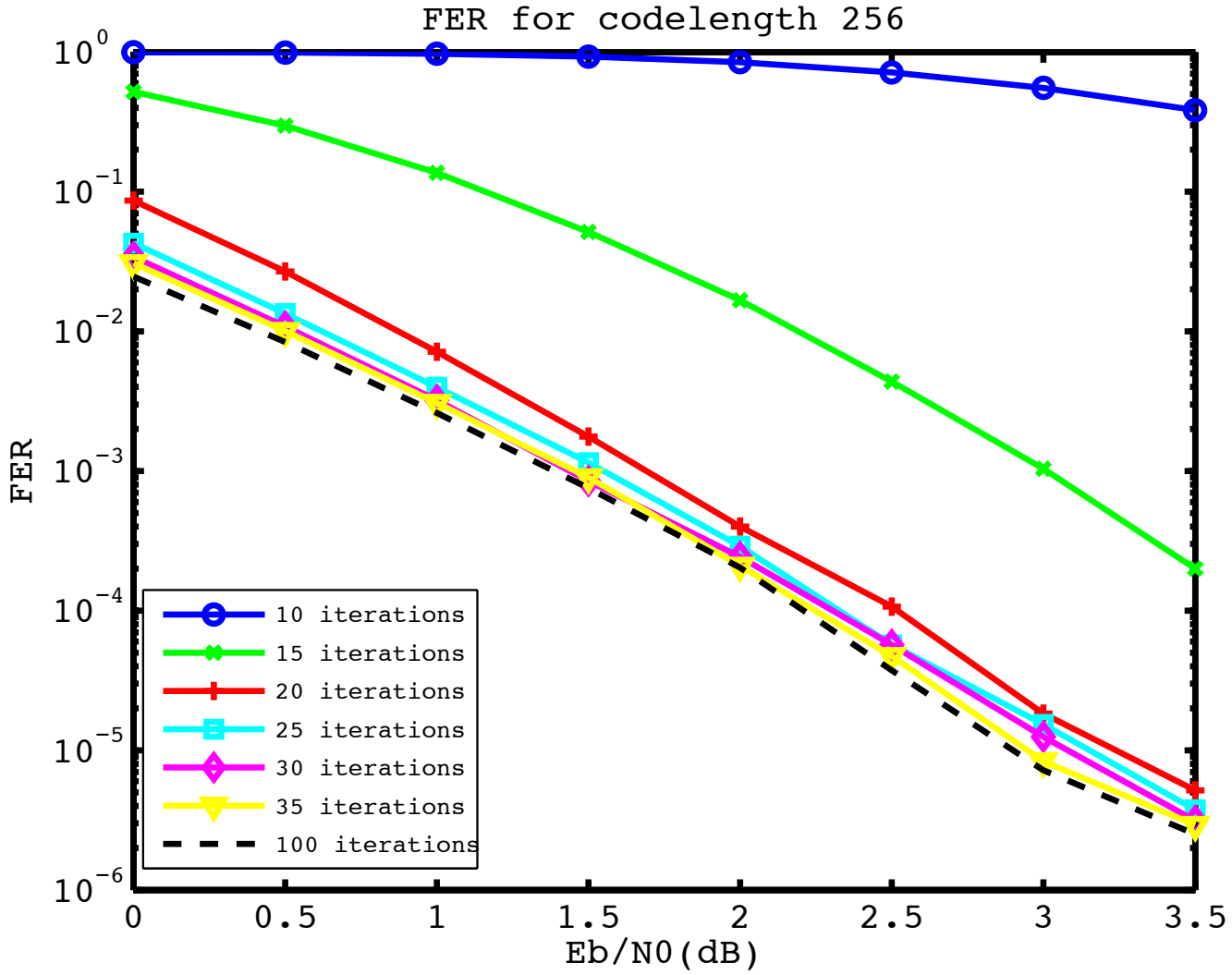
Results

Table : Speedup for 1024 length codeword(35 iterations)

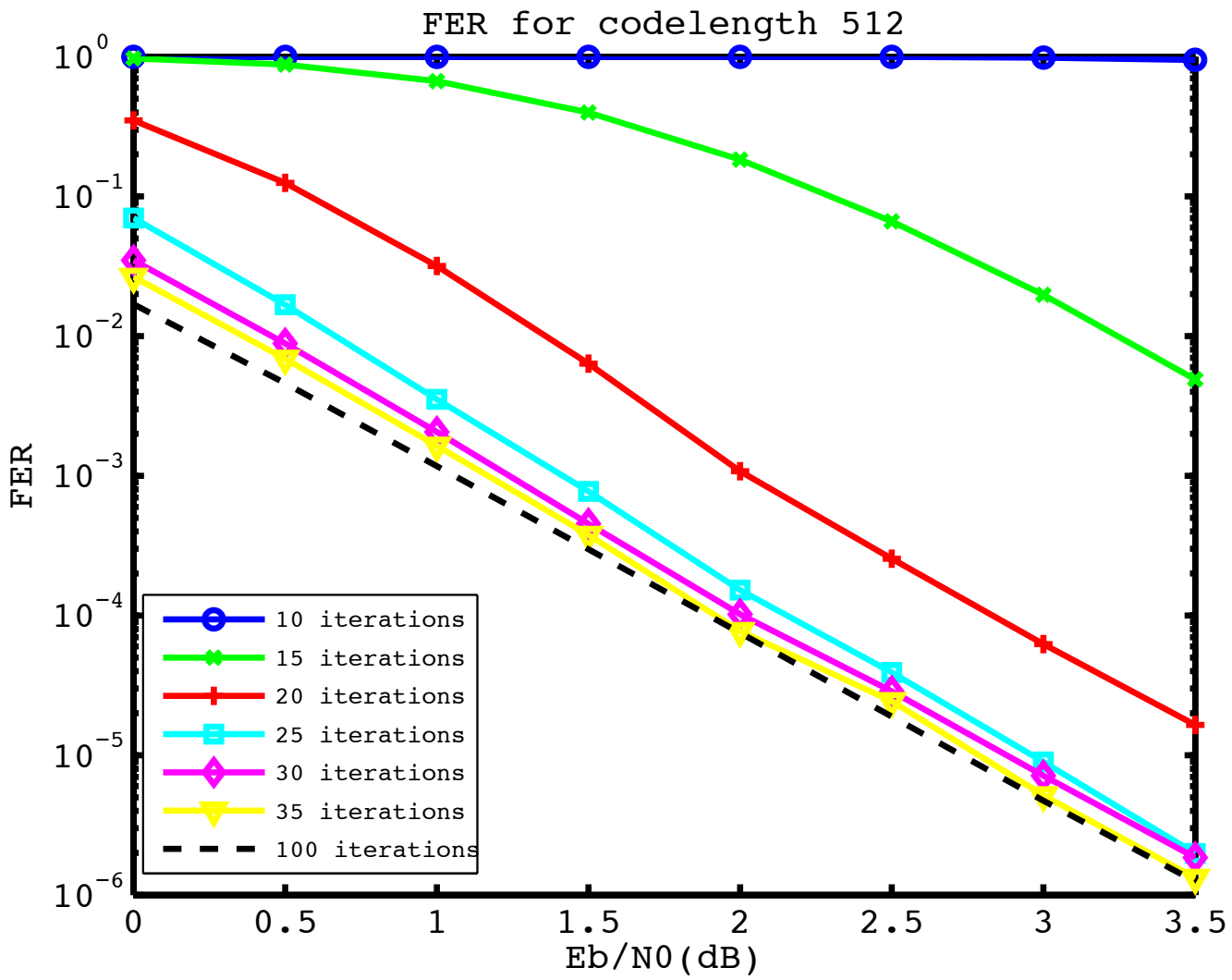
	CPU	CUDA
Platform	Single Intel Core	GTX 560 Ti
Time	0.06sec/codeword	0.038sec/288codewords
Throughput	8.33Kbps [2]	3.55Mbps
Speedup	-	436x

[2] The CPU code was executed on a single core using the general compiler optimizations. -O3 flag was used

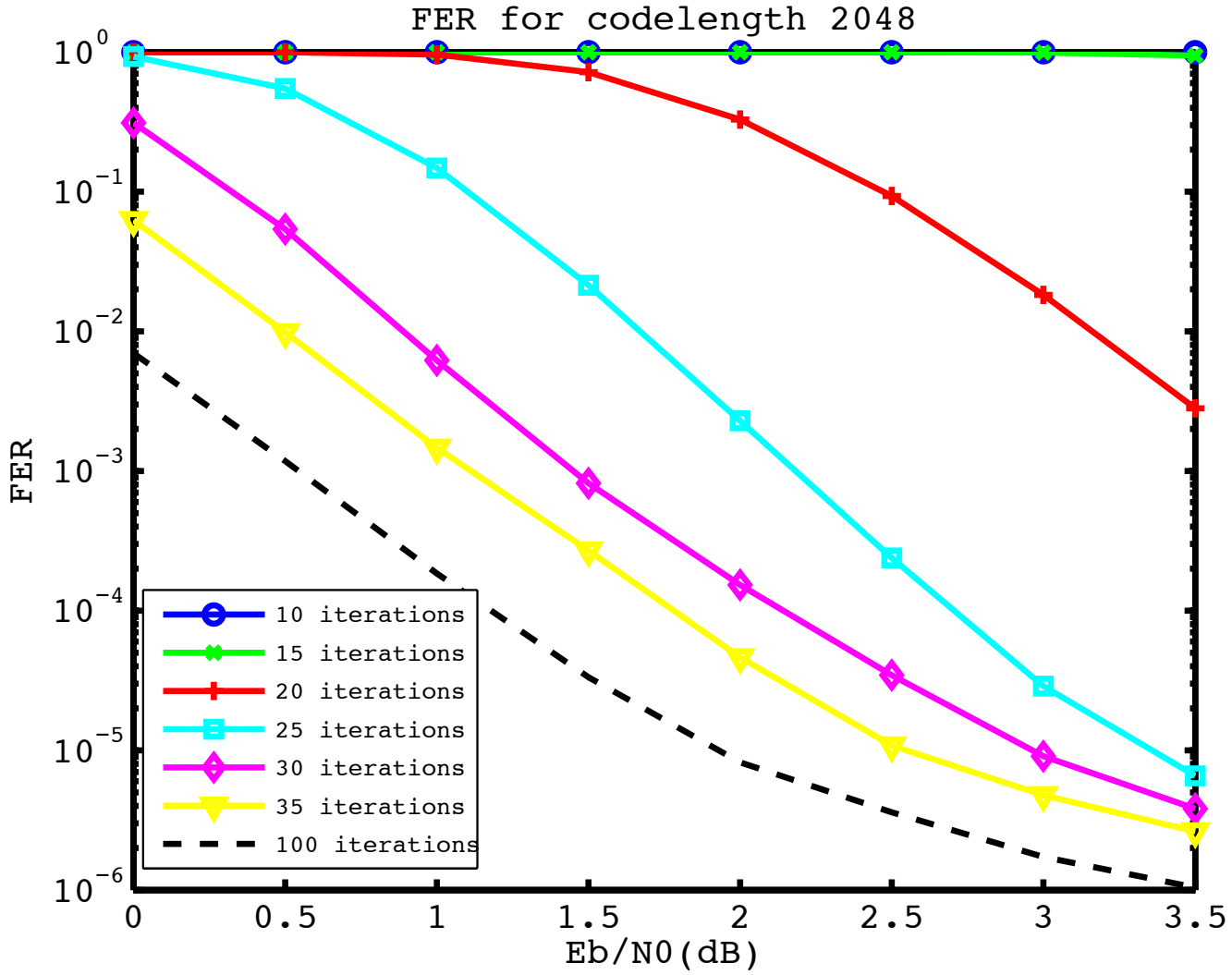
Results


FER for codelength 256
FER
Eb/N0(dB)
10 iterations
15 iterations
20 iterations
25 iterations
30 iterations
35 iterations
100 iterations

Results


FER for codelength 512
FER
Eb/N0(dB)
10 iterations
15 iterations
20 iterations
25 iterations
30 iterations
35 iterations
100 iterations

Results


FER for codelength 2048
FER
Eb/N0(dB)
10 iterations
15 iterations
20 iterations
25 iterations
30 iterations
35 iterations
100 iterations