Parallelization Techniques for Implementing Trellis Algorithms on Graphics Processors

Q. Zheng*, Y. Chen*, R. Dreslinski*, C. Chakrabarti†, A. Anastasopoulos*, S. Mahlke* and T. Mudge*
*EECS Dept, University of Michigan, Ann Arbor
†School of ECEE, Arizona State University, Tempe

***Abstract*— In this paper, we study different schemes to parallelize trellis algorithms for efficient implementation on a GPU. We consider parallelization schemes at the packet-level, subblock-level and trellis-level to increase the number of threads in a GPU implementation. At the trellis-level, we consider state-level, forward-backward traversal and branch-metric parallelism. To evaluate the performance of the different schemes, an LTE uplink Turbo decoder is implemented on an NVIDIA GTX470 GPU. Tradeoffs between throughput, latency and bit error rate are presented. Our most balanced configuration is simultaneously processing multiple subblocks in a packet in conjunction with recovery schemes and trellis-level parallelism, which can achieve a throughput of 19.65 Mbps with a latency of 0.56 ms at bit error rate of 10^{-5} for 1.3 dB channel SNR. We also show how different combinations of parallelization schemes can be used to satisfy systems with widely varying requirements of throughput, latency and bit error rate.**

I. Introduction

The trellis is a widely used graph in coding theory that describes the progression of symbols within a code. There are many popular trellis algorithms, including Viterbi algorithm [1], Baum-Welch algorithm [2], BCJR algorithm [3], etc. These algorithms are used in many systems, such as in speech recognition, communication protocols and data compression. In order to meet the timing deadlines of such systems, high throughput implementations of trellis algorithms are required. For example, the Viterbi algorithm that is used as the convolutional code in the WCMDA wireless protocol requires a 2Mbps decoding throughput in the downlink.

In this paper, we investigate the use of graphics processors (GPUs) for this application domain. They provide high throughput with good programmability and are very attractive for their raw compute power per dollar. Unfortunately, trellis algorithms do not map well onto these platforms due to relatively high computational dependencies. In spite of the imperfect match, there has been a growing interest in exploring whether GPUs have a role in that domain. GPU implementations have been proposed for trellis-based algorithms, such as those used in Turbo decoding [4]–[6] and Viterbi decoding [7]. In these papers, packet-level, subblock-level and state-level parallelization schemes were implemented to get high throughput. However, none of them considered the processing latency, and the tradeoffs between throughput, latency and bit error rate.

In this work, we study different parallelization techniques such as those at the packet-level, subblock-level and trellis-level. At the trellis-level, we consider state-level, forward-backward traversal and branch-metric parallelism. While trellis-level, subblock-level and packet-level parallelism all improve the throughput, trellis-level parallelism does not affect the latency or bit error rate. For the LTE uplink, we find that use of subblock-level parallelism with 256 subblocks in combination with state-level and forward-backward traversal at the trellis-level, and recovery schemes such as next iteration initialization and training sequences exceeds the LTE throughput requirement of 16.67 Mbps for bit error rate of 10^{-5} when implemented on the NVIDIA GTX470 GPU. If multiple packets are processed, our system can achieve a throughput of 29 Mbps that is comparable with [4] but with significantly lower packet latency.

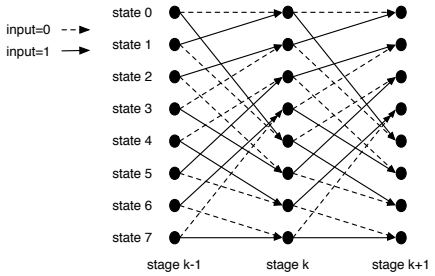

Fig. 1. Trellis structure of Turbo codes used in LTE

II. Background

A. Trellis Algorithms

The trellis is a graph representation of the state transitions of a finite state machine (FSM) for all possible input sequences. Fig. 1 is a typical structure of a trellis. Each column is a unit of time called a stage and each node represents a possible FSM state at each stage. A branch between two states corresponds to a possible state transition, depending on the input to the FSM. In the forward direction, the forward metric of a state s_k at stage k is the maximum (over all possible transitions from s_{k-1} to s_k) of the sum of the forward metric of states s_{k-1} and the branch metric corresponding to the transition from s_{k-1} to s_k. Similarly for the backward direction. After computing the forward and backward state metrics, the metric of each possible transition at stage k is evaluated as the sum of the forward metric of the starting state s_{k-1}, the branch metric corresponding to the transition from s_{k-1} to s_k and the backward state metric of s_k. Finally at each stage k, the metric corresponding to each input bit is evaluated as the maximum among all transition metrics corresponding to the same input bit. In Fig. 1, for example, the metric for bit 0

978-1-4673-5762-3/13/$31.00 ©2013 IEEE

is the maximum metric of eight transition metrics represented with dashed lines.

B. Performance Challenges on a GPU

A GPU is a programmable processor providing GFLOPS-level throughput, which makes it a good platform for computation intensive applications. However, it is a challenge to implement an algorithm that makes full use of the GPU resources. There are two main causes of underutilization: pipeline stall and thread inadequacy. Pipeline stall occurs when dispatch units fail to issue an instruction, mainly due to long memory access. Thread inadequacy happens if the number of thread blocks is smaller than that of streaming multiprocessors (SMs), or the number of threads in each thread block is not a multiple of thread context size, 32 for our case. To keep all the cores of a GPU active, an adequate amount of workload must be created. The parallelization schemes proposed here help create such a workload.

The memory system consists of on-chip memory, off-chip L2 cache and external memory. On-chip memory can be configured as either $48KB/16KB$ or $16KB/48KB$ shared memory/L1 cache. Shared memory is software managed, so when shared memory usage per thread is fixed, more shared memory leads to more threads and the GPU is better utilized. However, this also leads to smaller L1 cache and thus longer access time.

III. Parallelization Schemes

We consider three levels of parallelism: packet-level, subblock-level and trellis-level for higher GPU utilization.

A. Packet-level Parallelism

A packet is a formatted unit of data in a computer or communication network. In a GPU implementation, the input packets can be stored in a buffer so that they can be processed in parallel. The disadvantage of packet-level parallelism is that it results in long latency especially for the first packet in the buffer. This impairs the quality of service of time-constrained applications. The number of threads in a packet-level parallelism scheme is proportional to the number of packets that are processed in parallel.

B. Subblock-level Parallelism

A packet can be divided into several subblocks, which are processed in parallel. While this increases the number of threads, it leads to higher bit and packet error rates since the computations in each of the subblocks are not really independent from each other. Specifically, the computation of the ith subblock depends on the computations in the last stage of the $(i-1)$th subblock. Thus, if subblocks are processed in parallel, the initial values of latter subblocks are incorrect resulting in higher output error rate. One way to compensate for this performance loss is by employing recovery algorithms, e.g., training sequence (TS) and next iteration initialization (NII) [6].

In the TS algorithm, additional computations are done on the $(i-1)$th subblock to generate the dummy initial values of the ith subblock. The longer the training sequence, the larger is the number of additional computations and lower is the bit error rate (BER).

In the NII algorithm, the outputs of the $(i-1)$th subblock in the previous iteration are used as the initial values of the ith subblock in the current iteration. The idea behind NII is that the results of each iteration converge closer to the correct values than those of previous iterations. From an implementation perspective, TS requires additional operations, and NII needs additional memory.

C. Trellis-level Parallelism

There are three types of trellis-level parallelism. The first is state-level parallelism, in which the nodes in a stage is processed in parallel. There are no computational dependencies among the nodes in a stage and the processing of a node only depends on the nodes that are connected to it in the adjacent stages. State-level parallelism does not affect BER, and the number of threads due to state-level parallelism is proportional to the number of states in a stage.

TABLE I
SUMMARY OF PARALLELIZATION SCHEMES

Scheme	Throughput	Latency	Bit Error Rate
Packet-level	Better	Worse	No_Change
Subblock-level	Better	No_Change	Worse
Trellis-level	Better	No_Change	No_Change
Subblock+NII	Worse	No_Change	Better
Subblock+TS	Worse	No_Change	Better

The second type of parallelism is forward-backward traversal where the values are propagated in both forward and backward directions, and the propagations are independent. Forward-backward traversal (FB) results in more complex index and memory address computations, because two propagations must be separated during the calculation. Therefore, more instructions are executed to support FB parallelism, thereby lowering the throughput.

The third type of parallelism is branch-metric parallelism (BM), where the branches from a node in stage k to others in stage $k+1$ are processed in parallel. This is not as effective since the vector reduction parts cannot be parallelized. However for higher radix trellis that is obtained by combining multiple stages together, more threads can be generated from BM. Also less memory is used in this case. Overall, two threads are created in FB and the number of threads created by BM is equal to the radix degree.

Table I summarizes the parallelization schemes. From this table, we see that trellis-level parallelism improves throughput without impairing latency and BER. Packet-level and subblock-level parallelism improve throughput at the cost of either longer latency or higher BER. Both recovery schemes degrade the throughput but improve BER performance compared to only subblock-level parallelism.

IV. Results

A. Experimental Framework

We implemented a representative trellis algorithm, the BCJR algorithm [3], on the NVIDIA GTX470 GPU to evaluate

the performance of the different parallelization schemes. The GTX470 is based on Fermi architecture [8]. It can support at most 448 threads running at a time. It has a 64KB on-chip memory, a 768KB L2 cache and 1280MB external memory. We chose the BCJR algorithm as a case study since it is used in the Turbo code in LTE. The corresponding trellis structure has 8 states in a stage and is the same as shown in Fig. 1; however, the values propagate through the trellis in both directions. The LTE Turbo code configuration is used in our simulations: the packet size is 6144 bits and the code rate is 1/3 [9]. The baseline implementation is a sequential one (without any parallelization) with 0.0178 Mbps throughput and 345ms packet latency.

B. Experimental Results

We implemented two GPU memory configurations ($48KB/16KB$ and $16KB/48KB$ shared memory/L1 cache) with state-level, subblock-level and packet-level parallelism. We varied the number of subblocks from 1 to 512, and the number of packets from 1 to 84. We found that a larger L1 cache results in better timing performance. For instance, for the configuration with small input size (1 packet) and 64 subblocks, the larger L1 cache configuration achieves 30.8% higher throughput compared with smaller L1 cache configuration. So, we used the 48K L1 cache configuration in the rest of the experiments.

First, we studied the performance of different trellis-level parallelism schemes for different packet latencies. We use packet buffering latency as a metric to represent packet-level parallelism since it is a function of the number of packets being processed in parallel. Fig. 2 shows the performance of the different schemes when the subblock size is the same as the packet size. As the number of packets increases, the throughput increases. However, the throughput gains slow down when the number of packets is quite large. This is because the GPU is fully loaded and having more threads is not beneficial any more. The throughput improves quite a bit when state-level parallelism is combined with either FB or BM parallelism. Compared with the baseline scheme, state-level, state-level+FB and state-level+BM achieve a speedup of 5.1x, 7.4x and 5.8x, respectively. All schemes achieve a BER of 10^{-5} when SNR is 1.0 dB.

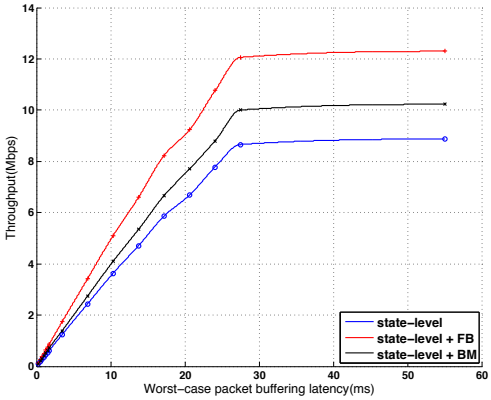

Fig. 2. Throughputs and latencies of different trellis-level parallelization schemes when the packet size is the same as the subblock size and multiple packets are processed in parallel.

Next, we fixed the latency by considering only one packet and studied the effect of different numbers of subblocks and recovery schemes such as TS and NII. Fig. 3 and Fig. 4 show the throughput and SNR requirement for the different schemes. The length of a packet is 6144 bits, which equals the product of the subblock length and the number of subblocks. The SNR requirement presented here is the lowest value to achieve the given bit error rate of 10^{-5}. The SNR requirement of the baseline scheme is 0.9 dB. From this figure, we derive the following conclusions. 1) Increasing the number of subblocks provides higher throughput due to more parallelism, but has higher SNR requirement due to wrong initial values and shorter subblocks. 2) Longer training sequences have lower SNR requirement but lower throughput due to additional calculations. The SNR requirement saturates when the training sequence is long. For instance, TS-12 has almost the same SNR requirement as TS-full in which the training sequence is as long as a subblock. Additional computational overhead due to recovery schemes does not affect the throughput as much because of the high computational power provided by GPU. 3) Among the recovery schemes, the combination of NII and TS is the best. The scheme NII+TS-4 has nearly the lowest SNR requirement with a throughput of 4.26 Mbps when 512 subblocks are used per packet. Its throughput is comparable with that of NII or TS-4, but it has a lower SNR requirement.

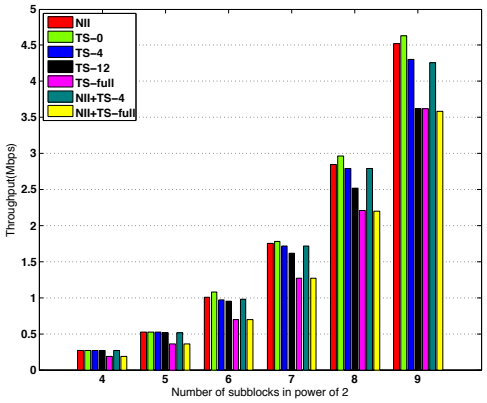

Fig. 3. Throughputs of schemes with different number of subblocks (per packet) and recovery schemes for bit error rate of 10^{-5}.

We also study the effect of increasing the radix of the trellis algorithm. Radix-4, which is derived by combining two stages into one, helps to double threads from BM compared with radix-2, and reduces the required memory because there is only half the number of stages. However, since twice the number of threads are generated, the amount of work each thread undertakes is still the same and the total amount of work done in a radix-4 implementation is two times that of radix-2. So, radix-4 is useful only when GPU is not fully loaded and the benefits from compactness outperform the overhead of redundancy. Our experiment shows that radix-4 outperforms radix-2 when the packet number ≤ 4.

C. Implementation Tradeoff

Different systems have different requirements for throughput, latency and BER. For instance, real-time gaming requires

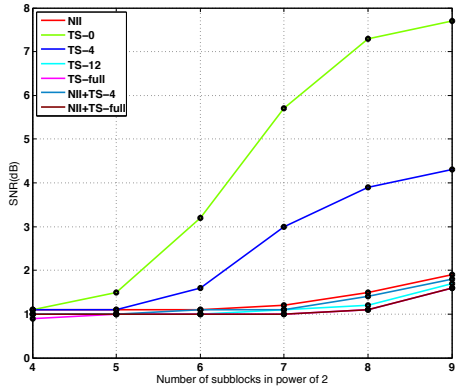

Fig. 4. SNR requirement of schemes with different number of subblocks (per packet) and recovery schemes for bit error rate of 10^{-5}.

low latency but medium throughput and BER; TCP-based service requires both low BER and high throughput but can tolerate long latency. In our study, we combined different schemes to determine which combinations of parallelism were suitable for which applications. First, we found that subblock parallelism with a combination of NII and short TS (NII+TS-4) achieves the best tradeoff between bit error rate and throughput. Next, we implemented NII+TS-4 with packet-level and trellis-level parallelization schemes to meet the 16.67Mbps LTE uplink throughput. Table II shows the throughput, latency, SNR requirement and BER of the different schemes. Note that trellis-level parallelism improves the throughput significantly and should be used at all times (row 2). If SNR requirement is low, subblock-level parallelism has to be used with caution and trellis-level and packet-level parallelisms are better options (rows 3 and 4). If the system has a rigid latency constraint, trellis-level or subblock-level parallelisms should be used to achieve high throughput with low latency (rows 2 and 4).

Table III compares the performance of our scheme with other LTE Turbo decoder implementations on a GPU. For a fair comparison, we scaled the throughputs in [5] and [6] by the processor frequency and the number of processors in the GPU. Our scheme achieves the best throughput with good BER. While [4] has comparable throughput, it requires processing 50 packets to achieve 27.5 Mbps. In comparison, our scheme needs to process 10 packets to achieve 29.0 Mbps throughput, resulting in significant reduction in the worst-case packet latency.

TABLE II

IMPLEMENTATION TRADEOFF

Schemes			TH*	WPL*	SNR*	BER*
TL+	Subblock Num	Packet Num	(Mbps)	(ms)	(dB)	
-	512	1	4.26	1.44	1.7	1.6×10^{-3}
SL+	512	1	20.49	0.55	1.7	1.6×10^{-3}
SL	256	2	21.09	1.07	1.3	4.1×10^{-4}
SL,FB+	256	1	19.65	0.56	1.3	4.1×10^{-4}
SL,FB	128	10	29.00	4.58	1.1	2.0×10^{-4}

* TH = Throughput, WPL = Worst-case Packet Latency, SNR = Signal-to-Noise Ratio requirement, which is the lowest value to achieve BER of 10^{-5}, BER = Bit Error Rate when SNR = 1.0 dB

+ TL = Trellis-level parallelism, SL = State-level parallelism, FB = Forward-Backward traversal

TABLE III

PERFORMANCE COMPARISON

Work	GPU	Original Throughput (Mbps)	Scaled Throughput (Mbps)	BER*
[5]	Tesla C1060	2.1	3.77	1.0×10^{-2}
[6]	GeForce 9800	2.4	3.50	1.0×10^{-4}
[4]	GTX 470	27.5	27.5	Not known
Ours	GTX 470	29.0	29.0	2.0×10^{-4}

* BER here is the bit error rate when SNR = 1.0 dB

V. CONCLUSION

We implemented different schemes to parallelize trellis algorithms for GPU implementations. These include packet-level, subblock-level and trellis-level parallelism. These schemes are general and can be used in other parallel computing platforms. We considered the effect of on-chip memory configuration of the GPU and found that a larger L1 cache shows better throughput performance compared to one with a smaller L1 cache but a larger shared memory. When multiple subblocks (256) are processed in parallel, the combination of NII and TS algorithms with a 4-bit training sequence has the best performance in terms of throughput and BER. However, to achieve the LTE uplink throughput, a configuration with trellis-level parallelism in combination with subblock-level parallelism can achieve a throughput of 19.65 Mbps with latency of 0.56 ms at BER of 10^{-5}. We also show how different combinations of parallelization schemes can be used to satisfy systems with widely varying requirements of throughput, latency and bit error rate.

VI. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their useful comments and suggestions. We also thank ARM Ltd, who supported this work.

REFERENCES

[1] G. D. Forney, Jr., "The Viterbi Algorithm," in *Proc. of IEEE*, vol. 61, Mar. 1973, pp. 268–278.

[2] L. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics*, vol. 41, pp. 164–171, Feb. 1970.

[3] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Intell. Transport. Syst.*, vol. 20, pp. 284–287, Mar. 1974.

[4] M. Wu, Y. Sun, G.Wang, and J. Cavallaro, "Implementation of a High Throughput 3GPP Turbo Decoder on GPU," *Journal of Signal Processing Systems*, vol. 65, pp. 171–183, 2011.

[5] D. Lee, M. Wolf, and H. Kim, "Design space exploration of the Turbo decoding algorithm on GPUs," in *CASES'10*, 2010, pp. 217–226.

[6] D. Yoge and N. Chandrachoodan, "GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications," in *25th International Conference on VLSI Design*, Jan. 2012, pp. 149–154.

[7] D. Zhang, R. Zhao, L. Han, T. Wang, and J. Qu, "An Implementation of Viterbi Algorithm on GPU," in *International Conference on Information Science and Engineering*, Dec. 2009, pp. 121–124.

[8] "Next Generation CUDA Compute Architecture: Fermi," White Paper, NVIDIA, 2009.

[9] *Technical Specification: 3GPP TS 36.212*.