Low Power Design of Turbo Decoder Module with Exploration of Energy-Performance Trade-offs

K.C. Shashidhar[†⋆] Arnout Vandecappelle[†] Francky Catthoor[†‡]
[†]IMEC vzw, Kapeldreef 75, B-3001 Leuven, Belgium.
{⋆CS,‡ESAT} Dept., Katholieke Universiteit, Leuven, Belgium.
{kodambal, vdcappel, catthoor}@imec.be

Abstract

Turbo coding has become an attractive scheme for design of current communication systems, providing near optimal bit error rates for data transmission at low signal to noise ratios. However, it is as yet unsuitable for use in mobile systems owing to the high energy consumption of current implementations of the scheme. Due to the data dominated nature of the decoder, a memory organization providing sufficient bandwidth is the main bottleneck for energy. We have developed the Data Transfer and Storage Exploration methodology to optimize and trade off energy consumption and performance. This paper discusses the exploration of the cycle budget versus energy trade-off for the turbo decoder module, which was obtained using our storage cycle budget distribution tool.

1 Introduction

Faithful replication of a transmitted signal at the receiver depends mainly on the performance limitation of the coding scheme employed. Most known traditional coding schemes have fallen short of the theoretical bound laid down by Shannon limit [1] on what best can be achieved. It's only recently that a new scheme, called *turbo coding* [2], has come close to the limit in providing near optimal bit error rates for data transmission at low signal to noise ratios. Since it outperforms all previously known forward error correction channel codes, the technique has naturally become an attractive choice for design of current communication systems.

Many flavors of the turbo coding scheme can be seen in recent literature. A general scheme would be as shown in Figure 1. The information bit stream I, split into *frames*, is encoded by the convolutional

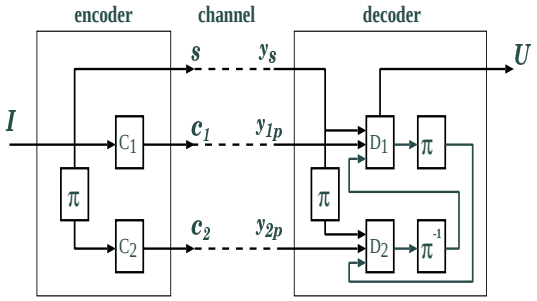

Figure 1: Turbo coding-decoding scheme

encoders C_1 and C_2. C_2 encodes the interleaved sequence of I. The encoded signals are transmitted over the channel, where noise gets added on to the signals. At the receiver, the signals are fed to the decoders D_1 and D_2. The decoding within a single decoder can be done in several ways [3, 4]. In our implementation, the decoding is done using the Maximum-A-Posteriori (MAP) algorithm [3]. The decoder is essentially an iterative scheme, wherein, the decoded signal from one decoder is fed to the other decoder and the process repeated until the required degree of convergence is achieved.

The main drawback of the turbo coding scheme, notwithstanding its near optimal performance, is the large latency and energy consumption of the decoder. This is due to the large number of memory accesses of the highly data dominated, iterative decoding scheme. Hence, memory organization is a very critical requirement for an efficient implementation of the scheme. Data Transfer and Storage Exploration (DTSE) [5] is a systematic methodology for optimization of memory accesses in such a data dominated application.

Issues relating to VLSI implementation of the turbo coding scheme have been addressed in [6, 7] and low complexity implementations to reduce en-

ergy consumption have been provided in [8, 9, 10], and many others. But, memory organization and optimization issues are not sufficiently addressed in literature. Memory access optimization, by applying early transformation steps of the DTSE methodology, has been investigated and important gains thereof have been shown in [11, 12]. However, the latter result is a single design point. The global energy and performance trade-off possibilities with respect to memory bandwidth, which are crucial for a designer to make a sound choice of the memory architecture, are as yet not available. In this work, our storage bandwidth optimization tool is demonstrated to provide such trade-off points in an implementation of the turbo decoder module.

In contrast to traditional design tools, which produce a *single optimal* solution, this trade-off curve allows the designer to explore the design space. For the turbo decoder, latency and throughput can be improved by sacrificing some energy consumption, or conversely more energy can be saved by lowering the performance. The designer can only efficiently decide on these trade-offs if there is a way to visualize them. Therefore, we propose pareto plots, which are generated automatically, to show the energy cost of achieving a particular throughput.

Outline of the paper: Section 2 briefly introduces the DTSE methodology and positions our work. Section 3 discusses the applied step in detail on the turbo decoder implementation. Section 4 discusses the results obtained, their interpretation and advantages. Section 5 concludes the article.

2 Data Transfer and Storage Exploration Methodology

In data dominated applications, typically found in the multi-media and telecom domain, data storage and transfers are *the* most important factors in terms of energy consumption, area and system performance. DTSE is a systematic, step-wise, system-level methodology to optimize data dominated applications for memory accesses, and hence, energy consumption [5]. The main goal of the methodology is to start from the source code specification of the application (for e.g. in the C language) and determine an optimal execution order for data transfers, together with an optimal memory architecture for data storage.

The DTSE methodology is explained at length along with case studies in [5], but to position our work, we briefly summarize it here. It essentially consists of the following *orthogonal* steps that are sequentially applied:

1. **Global Data Flow Transformations**: Reduces redundant copies of data and removes data flow bottlenecks.
2. **Global Loop Transformations**: Reduces system level buffers required because of long delays between production and consumption of intermediate values.
3. **Data Reuse Decisions**: Exploits hierarchical memory organization to make use of available temporal locality in the data accesses.
4. **Storage Cycle Budget Distribution**: Distributes available cycle budget over the iterative parts of the specification in a balanced way globally, such that the required memory bandwidth is reduced.
5. **Memory Allocation and Assignment**: Determines the optimal memory architecture for the data and the required assignment.
6. **In-place Optimization**: Finds the optimal placement of data in the memories such that the required memory size is minimal and also removes cache conflict misses.

The first 3 steps are essentially platform independent source code transformation steps which have previously been applied on the turbo decoder. The results are available in the literature [11, 12]. This paper completes the script with the memory organization steps 4 and 5. These steps make it possible to identify a range of energy-performance trade-off possibilities. Step 6, as mentioned, optimizes the area consumption and the cache misses, but these are not bottlenecks in our implementation.

The trade-off points identified are represented as a *pareto curve* between energy consumption in memory and the storage cycle budget, as shown in Figure 2, for a given choice of memory architecture. The cycle budget ranges from fully sequential, where all memory accesses are scheduled one after the other, to the critical path schedule, where memory accesses are sequential only if there is a dependency between them. A smaller cycle budget means more parallel accesses with lesser freedom in ordering and hence higher energy consumption in the more complex memory required. With such a pareto curve, the designer can ignore the points which are clearly sub-optimal and concentrate on the interesting trade-off points.

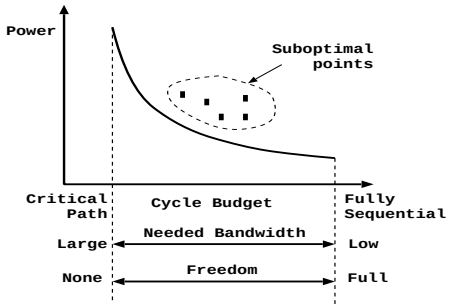

Figure 2: energy-cycle budget trade-off

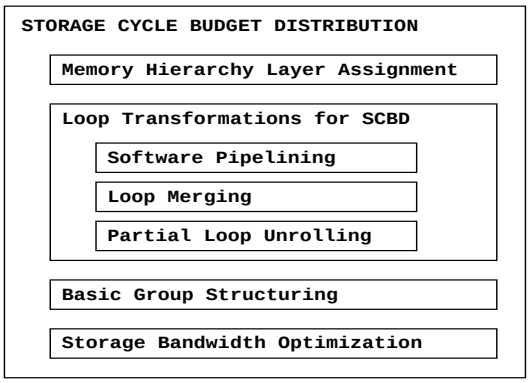

Figure 3: Sub-steps within the SCBD step

3 Storage Cycle Budget Distribution

In the DTSE methodology, the memory architecture is optimized before doing the detailed scheduling and synthesis. This implies that sufficient memory bandwidth is made available for a final schedule, which meets the required cycle budget of the application, but is still optimal in terms of energy consumption. This requires exploration along the length of the memory access schedule, and the Storage Cycle Budget Distribution (SCBD) step entails a tool supported methodology to achieve this. The following definitions are used further on:

- **Storage Cycle**: An abstract unit of time required to schedule transfers between the data paths and the memories, and between memories on different layers. It defines the granularity at which such transfers can be positioned in time.
- **Storage Bandwidth**: Total number of ports in the chosen memory architecture.
- **Basic Group**: A group of data that is together allocated and assigned to the same memory. The grouping is done to alleviate the complexity in dealing with each data item separately. Coarsely, each array is also termed a basic group.
- **Extended Conflict Graph** (ECG): Conflict graph which includes hyper edges and R/W/RW annotation for each edge, representing the maximum reads, writes and total memory accesses that can occur simultaneously. This additional information is required for capturing the complete conflict information when multi-port memories are available. In our discussion, we assume this extension whenever we refer to conflict graphs.

Given the control/data flow graph (CDFG) representing the behavior of the application, SCBD mainly determines, from the data dependencies, for which data, parallel access should be possible in order that cycle budget is met with minimum bandwidth requirements. This information is made available in the generated conflict graph for each cycle budget, from fully sequential to critical path budget, which the subsequent Memory Allocation and Assignment (MAA) step matches with a memory architecture, avoiding the conflicts. A valid final schedule can exist only if a memory architecture satisfying the conflict graph is found.

SCBD consists of sub-steps, shown in Figure 3. The Storage Bandwidth Optimization (SBO) is the main automated sub-step; the details about the implementation of the tool are available in [13, 14]. The efficiency of the resulting organization depends on the code transformations applied manually in the earlier sub-steps. Here we briefly discuss those sub-steps which are applicable for the turbo decoder implementation at hand.

3.1 Memory Hierarchy Layer Assignment

This is the first sub-step in SCBD, which decides for each basic group, the layer in the memory hierarchy it will be assigned to. The data reuse step, previously applied to the turbo decoder code [12], introduced five local buffers to make temporary copies of the reused data. The memory for these temporary copies is assigned a lower level in the hierarchy, so that, the costly duplicate accesses to the background memory is replaced by cheaper accesses to the lower level memory. The subsequent MAA step determines the optimal memory architecture for each layer. In effect, this sub-step fully determines

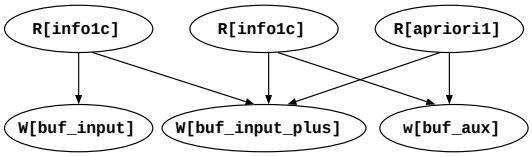

Figure 4: Dependence graph before sw. pipelining

Figure 5: Dependence graph after sw. pipelining

the static caching decisions.

3.2 Loop Transformations for SCBD

The SBO tool takes the access dependence flow graph as the input, therefore, the length of the critical path determines the scheduling flexibility. To increase the scheduling flexibility, the critical path length of the loop bodies has to be reduced. This is achieved by transformations on the loops like software pipelining, loop merging and partial loop unrolling.

On the turbo decoder module, software pipelining is applied on the 11 loops present, reducing the critical path length to a single cycle in each of them. Figure 4 shows the access dependence graph in a loop body, which requires 2 storage cycles in the critical path, as the data written to an array depends on data read from another array. This dependency is broken by delaying the write by one iteration and keeping the data alive in a register, reducing the critical path length to a single cycle as shown in Figure 5. This significantly reduces the storage cycle count of the module.

3.3 Basic Group Structuring

Basic group structuring refers to merging and/or splitting of basic groups in order to reduce the conflicts in the memory accesses. Merging of basic groups is done when elements in two different basic groups are always read and written together. Splitting of a basic group is done if it results in the removal of self-conflicts. Typically, splitting is done along separate dimensions.

In the turbo decoder, we split two basic groups in order to remove self-conflicts. In fact, without splitting, software pipelining cannot achieve the critical path in a single cycle for the loop body shown in Figure 4, as there are 2 reads from the same array, but on different dimensions.

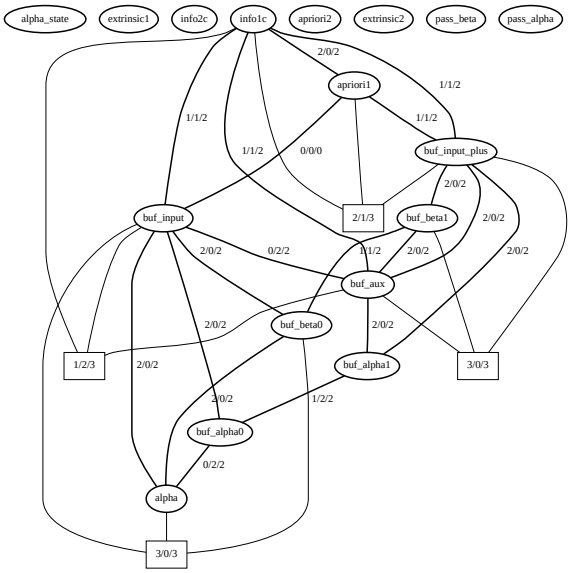

Figure 6: ECG for a budget of 430848 cycles

Code Version	No. of schedules	No. of storage cycles	
		Critical Path	Fully Sequential
Before	23	316416	622656
After	39	286560	622656

Table 1: Enhanced range by SCBD transformations

3.4 Storage Bandwidth Optimization

SBO is fully automated and as with most optimizers, is the most time consuming step. It optimizes the required storage bandwidth for a given cycle budget by partially ordering the CDFG at the basic group level. The outputs are the conflict graphs, for each cycle budget, that contain information about which basic groups are accessed simultaneously and therefore have to be assigned to different single port memories or a multi-port memory. Figure 6 shows the conflict graph generated by the SBO tool for the turbo decoder module for a certain cycle budget. The square boxes represent hyper conflict edges.

Table 1 shows the difference the applied code transformations made in the output of the SBO for the turbo decoder code. The critical path budget has come down significantly and at the same time more schedules have become visible. The SBO output about the required number of cycles is also used as a feedback to the earlier transformations to check for the possibility of further improvement in the solution.

Basic Group	Size	BW	NA	MA
alpha	3264	7	78432	1
info1c	800	4	12896	2
info2c	800	4	13293	4
apriori1	400	6	8848	4
apriori2	400	6	8845	2
extrinsic1	400	6	4800	2
extrinsic2	400	6	5197	2
alpha_state	64	7	2368	2
buf_alpha0	64	7	69228	2
buf_alpha1	64	7	69888	3
buf_beta0	64	7	69792	3
buf_beta1	64	7	68256	4
pass_alpha	64	7	2112	2
pass_beta	64	7	2016	2
buf_aux	8	6	24192	4
buf_input	8	4	28992	RF
buf_input_plus	8	6	24192	RF

BW = Bit-width; MA = Memory Assignment;
NA = Number of accesses (Reads + Writes);
NP = Number of ports; RF = Register File;
Energy consumption is in μJ/frame;

Mem.	NP	Size	BW	NA	Energy
1	1	3264	7	78432	48.726
2	1	2256	7	107462	51.285
3	1	128	7	139680	21.364
4	2	1272	7	114589	40.753
RF	-	16	6	53184	3.389
Total					165.517

Table 2: MAA for a budget of 430848 storage cycles

4 Results

The SBO and MAA tools are used in conjunction to provide accurate energy and area values, for each cycle budget, for each input memory architecture, from fully sequential to the critical path schedule. These values, in our case only for energy, plotted against the cycle budget gives the pareto curve, which the designer can use to decide on the initial memory architecture. This architecture can be further explored for in-place optimization, if required.

In the results that follow, we have used the energy consumption values of a 0.35μm CMOS process of Alcatel Microelectronics. The choice of word lengths of the data values, which are necessary for the estimation, are based on simulations without significant degradation in functionality.

A snapshot output of the MAA tool for the conflict graph of Figure 6 is shown in Table 2. The de-

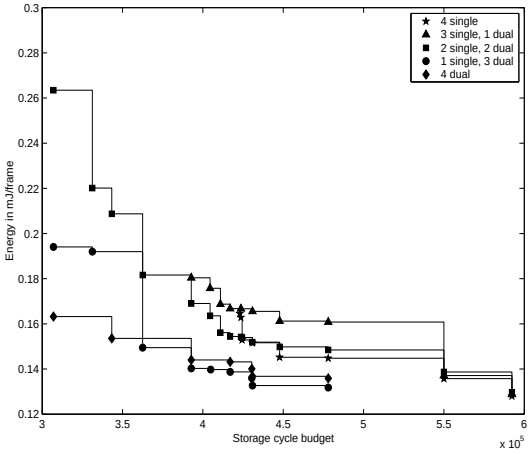

Figure 7: Pareto curves for 4 memories

cision of how many memories and ports to use is currently still manual. Therefore, for a given number of memories, the pareto curves are generated for the complete range of port configurations with single and dual-port memories. Figure 7, gives the pareto curves for the case of 4 memories. Depending on the required throughput, a port configuration with 4 single ports, with 3 dual and one single ports, or with 4 dual ports should be selected.

The trade-off points to be considered by the designer depends on the performance constraint that is being addressed. An increase in performance, as shown, also increases energy consumption. But, the available range of possibilities make a good trade-off feasible. When area is at a premium, though not in our context, a further energy-area trade-off becomes necessary.

It is observed that use of dual-port memories for small cycle budgets does not necessarily give poor energy values, when the size of memories are not too large, as in our application. In fact, if the performance requirements are high, dual port memories become a necessity because of unavoidable conflicts in accesses. At present, our power estimation model does not include the influence of interconnects on energy, but, the trends represented in the curves are expected not to differ much by its inclusion.

Figure 8 gives the pareto curves for different number of memories with all different port configurations summarized in a single pareto curve. These curves are obtained by generating pareto curves similar to the one shown for 4 memories in Figure 7 with other values for the number of memories and collecting the optimal points for different cycle budgets.

Based on the pareto curves for different memo-

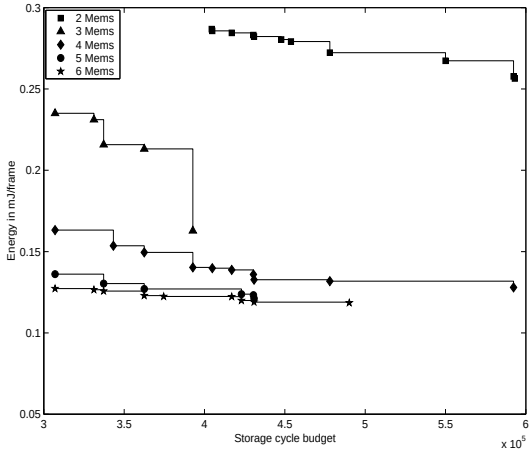

Figure 8: Pareto curves for different memories

ries, the cost of limiting the number of memories can be evaluated. A very large number of small memories is best avoided, as this increases the complexity for interconnection synthesis and layout. Figure 8 shows that for all cycle budgets, 5 memories are sufficient to reach acceptable energy consumption.

5 Conclusions

Identifying energy-performance trade-offs is crucial for an optimal implementation of turbo decoder. Since the decoder is highly data dominated, the maximum gain is in finding the energy optimal memory architecture which meets the performance requirements. We have identified the available trade-offs of an implementation of MAP decoding algorithm, using a systematic methodology. Tool support made possible an almost impossible task of manually checking the entire range of trade-offs for different combinations of memory architectures. Code transformations have been crucial to allow the SBO tool to provide an enhanced range of cycle budget distributions and hence, exposing the otherwise hidden trade-offs.

The methodology explained makes the trade-off points in the turbo decoder visible to the designer, very early in the design cycle, allowing better design exploration. This coupled with further optimization of memory accesses is an important step towards the possibility of using turbo coding schemes in a whole range of wireless communication applications.

References

[1] C.E. Shannon. *A Mathematical Theory of Communication*. Bell Sys. Tech. Journal, Vol. **27**, pp. 379-423,623-656. 1948.

[2] C. Berrou et. al. *Near Optimum Error Correcting, Coding and Decoding: Turbo Codes*. IEEE Trans. on Comm., Vol. **44**, No.10, pp. 1261-1271. 1996.

[3] L. R. Bahl et. al. *Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate*. IEEE Trans. on Info. Theory, Vol.**20**, pp. 284-287. 1974.

[4] C. Berrou et. al. *A Low Complexity Soft-Output Viterbi Decoder Architecture*. IEEE Intl. Conf. Record Comm., Geneva. 1993.

[5] F. Catthoor et. al. *Custom Memory Management Methodology - Exploration of Memory Organization for Embedded Multimedia System Design*. Kluwer Academic Publishers. 1998.

[6] G. Masera et. al. *VLSI Architectures for Turbo Codes*. IEEE Trans. on VLSI Systems, Vol. **7**, No.3, pp. 369 -379. 1999.

[7] Z. Wang et. al. *VLSI Implementation Issues of Turbo Decoder Design for Wireless Applications*. IEEE Workshop on Signal Proc. Sys.: Design and Implementation, Taipei. 1999.

[8] H. Dawid, H. Meyr. *Real-time Algorithms and VLSI Architectures for Soft Output MAP Convolutional Decoding*. IEEE Symp. on Personal, Indoor and Mobile Radio Comm., Vol.**1**, pp. 193-197. 1995.

[9] F. Gilbert et. al. *Low Power Implementation of a Turbo-Decoder on Programmable Architectures.* ASP-DAC, 2001.

[10] S. Hong *et. al. Design and Implementation of a Low Complexity VLSI Turbo-Code Decoder Architecture for Low Energy Mobile Wireless Communications.* Journal of VLSI Signal Processing, Vol. **24**, No.1, pp. 43 -57. 2000.

[11] C. Schurgers et. al. *Memory Optimization of MAP Turbo Decoder Algorithms*. IEEE Trans. on VLSI Systems, Vol. **9**, No.2, pp. 305-312. 2001.

[12] F. Maessen et. al. *Memory Power Reduction for the High-Speed Implementation of Turbo Codes*. Symp. on Vehicular Comm. and Tech., Belgium. 2000.

[13] E. Brockmeyer et. al. *Low Power Storage Cycle Budget Distribution Tool Support for Hierarchical Graphs*. 13th ISSS, Spain. 2000.

[14] E. Brockmeyer et. al. *Systematic Cycle Budget versus System Power Trade-off: A New Perspective on System Exploration of Real-time Data-dominated Applications*. ISLPED, Italy. 2000.