

Memory Sub-Banking Scheme for High Throughput MAP-Based SISO Decoders

Mayank Tiwari, Yuming Zhu, and Chaitali Chakrabarti

Abstract—The sliding window (SW) approach has been proposed as an effective means of reducing the memory requirements as well as the decoding latency of the maximum a posteriori (MAP) based soft-input soft-output (SISO) decoder in a Turbo decoder. In this paper, we present sub-banked memory implementations (both single port and dual port) of the SW SISO decoder that achieves high throughput, low decoding latency, and reduced memory energy consumption. Our contributions include derivation of the optimal memory sub-banked structure for different SW configurations, study of the relationship between memory size and energy consumption for different SW configurations and study of the effect of number of sub-banks on the throughput/decoding latency for a given SW configuration.

Index Terms—High throughput, memory sub-banking, sliding window (SW), tradeoffs, Turbo decoder.

I. INTRODUCTION

In recent years, Turbo codes have become very popular because of their near-optimal performance [1], and have been adopted in mobile standards such as 3GPP for IMT-2000 and wideband code division multiple access (WCDMA). The superior performance is due to a combination of parallel concatenated coding, iterative decoding, large interleaver size, etc. The large frame size of Turbo codes and the iterative decoding process results in large decoding latency. The decoding latency has to be reduced in order to make Turbo-based systems acceptable for real-time voice communication and other applications that require instant data processing, like hard disk storage and optical transmission.

The Turbo decoder consists of two *soft-input soft-output* (SISO) decoders and interleavers/de-interleavers; the decoding latency is a function of the interleaver latency and the SISO decoding latency. In order to reduce the decoding latency and increase the throughput of the SISO decoder, the sliding window (SW) approach has been used in [2]–[6]. A comprehensive study of the tradeoffs between area, energy, and throughput for different SW configurations was done in [3] for monolithic memory. A similar analysis of computational hardware and memory for SISO *a posteriori probability* (APP) algorithm using a tile graph was presented in [6].

Most of the existing work on Turbo decoder architectures assume a monolithic memory structure. However, memory sub-banking is an effective means of achieving high throughput as discussed in [4], [5], [7]. The implementation in [4] employed three unique data RAMs with read-modify-write access in single cycle, while the one in [5] employed two dual-port data RAMs. The discussions in [4], [5] were limited to the $\eta = 1$ case presented in this paper.

In this work, we derive a systematic way of generating sub-banked structures using standard RAMs for high throughput SW-based SISO decoders. We evaluate the structures with respect to area, throughput, and energy consumption and provide a tradeoff analysis between the different parameters. The main contributions are as follows.

- Derivation of the optimal single-port memory sub-banking structure (number and size of each sub-bank) that supports

very high throughput and low SISO decoding latency for a given SW configuration (corresponding to a specific value of η).

- Study of the relationship between number of sub-banks, memory size, throughput and memory energy for a given SW configuration.
- Study of throughput, number of sub-banks, memory size and memory energy for different SW configurations.

Such a comprehensive study is intended to aid the designer in choosing the optimal memory configuration given the constraints on memory size, number of sub-banks, throughput/decoding latency and energy consumption.

The rest of the paper is organized as follows. Section II gives a brief description of Turbo coders, *maximum a posteriori* (MAP) algorithm and application of SW on MAP-based SISO decoder. Section III derives the memory size and number of single-port and dual-port sub-banks for the proposed optimal sub-banking structure, followed by trade-offs between throughput, memory size, number of sub-banks and energy consumption for nonoptimal schemes. This section also provides comparisons with previous work. Section IV concludes the paper.

II. SISO DECODER ARCHITECTURE

A. Turbo Decoder Structure

A Turbo encoder and an iterative decoder are shown in Fig. 1. The Turbo encoder consists of two recursive systematic convolutional (RSC) encoders and an interleaver. The Turbo decoder consists of two SISO decoders (corresponding to the two RSC encoders), an interleaver (I) and a de-interleaver (I^{-1}) placed between the two decoders. The first SISO decoder generates soft outputs, which are interleaved and used to produce an improved estimate of the *a priori* probabilities of the information sequence for the second decoder. The output of the second SISO decoder is fed to the first SISO decoder through the de-interleaver. The SISO decoders are typically implemented using the MAP class of algorithms. The MAP algorithm estimates the most likely information bit in a coded sequence.

B. Map Algorithm

The MAP algorithm minimizes the symbol (or bit) error probability. For each transmitted symbol, it generates a soft output in the form of APP based on the received sequence. The log-likelihood ratio $\Lambda(c_t)$ can be computed as

$$\Lambda(c_t) \approx \max_{e \in B_t^1} \{ \alpha_{t-1}[S(e)] + \gamma_t(e) + \beta_t[E(e)] \} - \max_{e \in B_t^0} \{ \alpha_{t-1}[S(e)] + \gamma_t(e) + \beta_t[E(e)] \} \quad (1)$$

for $1 \leq t \leq N$, where N is the frame length, B_t^i is the set of trellis transitions e that are caused by input $c_t = i$, $i \in \{0, 1\}$, $S(e)$ and $E(e)$ are the start and end states of trellis transition e . α_t and β_t are the path metrics, and γ_t is the branch metrics at time t . It is shown in [1] that α can be computed with a forward recursion and β can be computed with a backward recursion. The function \max^* is defined as $\max^* \{x, y\} \triangleq \max \{x, y\} + \delta(x, y)$, where $\delta(x, y)$ is a correction factor [8].

C. Sliding Window Approach

In the standard MAP-based SISO decoder, the decoding latency is equal to the received frame size. For large frames, the memory required

Manuscript received May 16, 2004; revised September 30, 2004. This work was supported by CEINT at ASU, and by NSF-ITR under Grant 0325761.

The authors are with the Department of Electrical Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: yuming.zhu@asu.edu).

Digital Object Identifier 10.1109/TVLSI.2004.842937

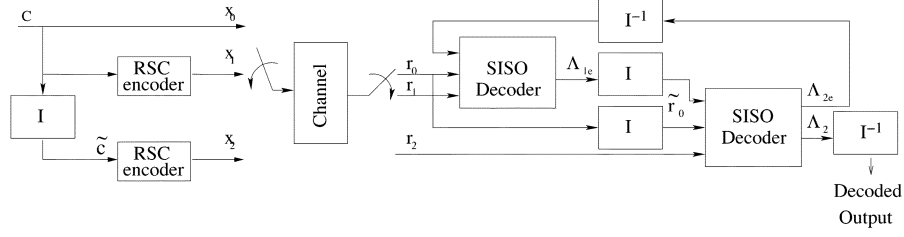
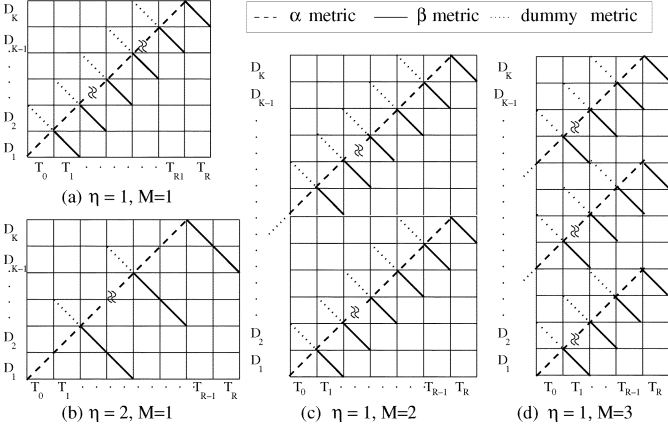


Fig. 1. Turbo encoder and iterative decoder. (Adopted from [1]).

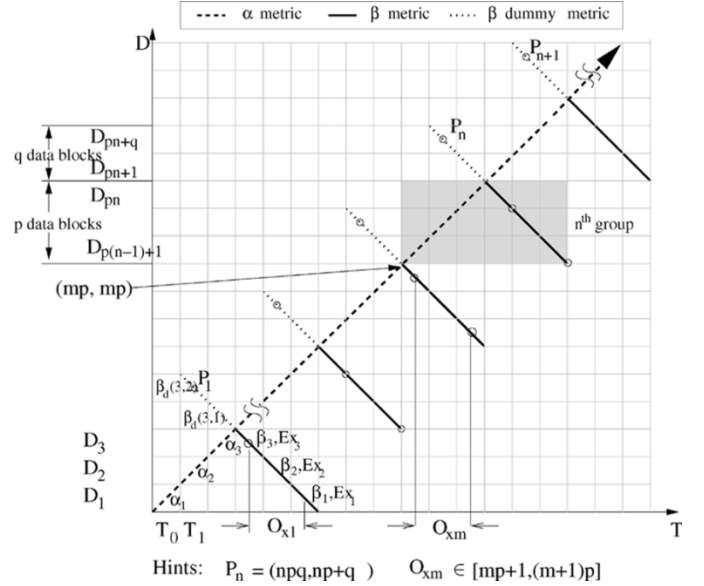

 Fig. 2. SW configurations. (a) and (b) $M = 1$. (c) $M = 2$. (d) $M = 3$ cases.

for the decoder is also excessive, since for a frame of size N , the decoder needs to store N survivors and N survivor path metrics during the forward and backward recursion calculations.

The SW approach partly eliminates the large storage problem of SISO decoders by initializing the α and/or β metrics at an intermediate stage. In fact, it can start at any stage, and after a depth of L stages, the decision is likely to be made as reliably as starting from the initial stage [3], [7]. L is typically 5 to 10 times the constraint length. The SW approach can be applied either (1) to α or β metric calculation, or (2) to both α and β metrics calculations.

In order to better explain the SW approach, we introduce a parameter η defined as the ratio of actual metrics calculated per dummy metric. Let $\eta = p/q$, where p and q are integers, and p/q is an irreducible rational number. Then, the number of stages of actual calculations for L dummy stages of calculations is ηL (by definition). Let the data frame be divided into blocks of size L/q . Then each frame consists of $K = qN/L$ data blocks. Let M be the number of blocks that are processed concurrently for high throughput applications. Concurrent processing is achieved by pipelining the computation units by M levels and processing the M blocks in an interleaved fashion. Fig. 2 describes different SW configurations, where each configuration is represented in a two-dimensional space with the vertical axis corresponding to the data blocks and the horizontal axis corresponding to time-slots (defined by the time required to process one data block). Fig. 2 also shows how β metrics calculations are done from an intermediate stage and α metrics calculations are done from the initial stage for $M = 1, 2, 3$.

The throughput of the MAP-based SISO decoder is defined by the number of bits processed, N , divided by the latency LA_{siso} , i.e., throughput = N/LA_{siso} . LA_{siso} is clearly dependent on the clock period, t_{clock} , which is again dependent on the delay in the recursive loops of the α , β , and dummy- β calculations. Since the computation units are pipelined to M levels, t_{clock} is inversely proportional to M . Apart from t_{clock} , and the number of outputs N , LA_{siso} is also a function of the latency due to use of pipelined computation units, Δ ,


 Fig. 3. SW configuration for $\eta = 3/2$, $M = 1$.

and the overhead due to data loading before beginning of α computation plus the time to complete the computation of the last block. For $\eta = p/q$, the overhead for each of the M blocks is $2q - 1$ time slots prior to start of the computation and p additional time slots to finish the computation. The latency of the decoder is then

$$LA_{\text{siso}} = \left[N + M(p + 2q - 1) \cdot \frac{L}{q} + \Delta \right] \cdot t_{\text{clock}} \\ = \left[N + ML \left(\eta + 2 - \frac{1}{q} \right) + \Delta \right] \cdot t_{\text{clock}}. \quad (2)$$

While LA_{siso} increases with increase of η and decreases with increase of M , the effect of η is not as significant as the effect of M . As M increases, t_{clock} reduces by a factor of M , and the overhead and Δ increases by a factor of M . Thus for large values of N , increasing M to increase the throughput is still the best option. However, in order that an output be generated in every clock cycle, the computation units have to be provided with a large volume of data. The subbanked memory structure described in Section III provides a mechanism to achieve the high memory bandwidth required for high throughput SISO decoding.

III. SUB-BANKED STRUCTURE FOR SISO DECODERS

For a given N , L , η and M , our aim is to design sub-banked memory configurations that are able to provide multiple data simultaneously (from memory) such that the computation units can each generate an output in every cycle ($= t_{\text{clock}}$). Assume that each sub-bank supports 1 access per cycle. The optimal memory configuration is then the sub-banked structure that meets this requirement with minimum total memory and minimum number of sub-banks. In this section, we provide a detailed analysis for the case $M = 1$. If $M > 1$, the optimal

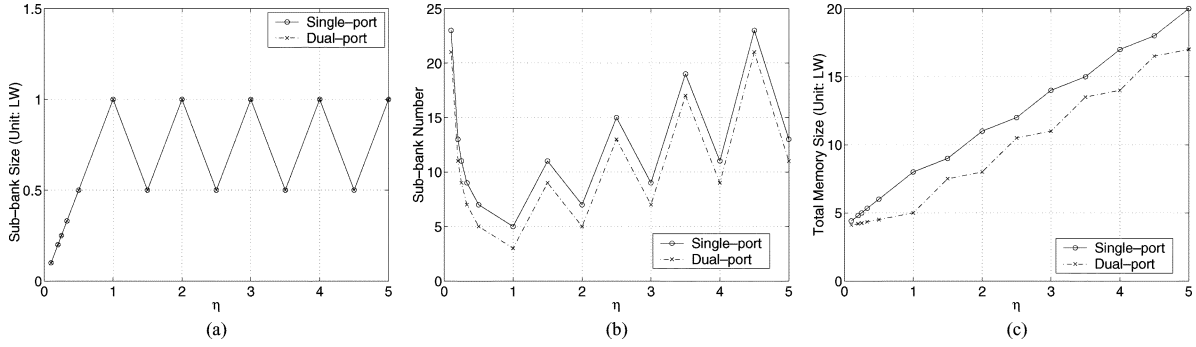


Fig. 4. Results for the optimal sub-banked configuration for $M = 1$ as a function of η . (a) Sub-bank size. (b) Number of sub-banks. (c) Total memory size.

number of sub-banks is the same as the $M = 1$ case, but the size of each sub-bank is M times larger.

A. Optimal Single-Port Sub-Banking for $\eta = p/q$, $M = 1$

In order to derive the optimal sub-banking structure, we make use of Fig. 3 and the following notation: Define α_i and β_i as the actual α and β metrics calculation for block D_i . Let $\beta_d(i, k)$ be the dummy β calculation needed to start β_i , where k varies from q to 1. $\beta_d(i, k)$ is performed on D_{i+k} . For the case when $q = 1$ (corresponding to $\eta = 1, 2, 3, \dots$), there is only one value of $k (= 1)$ and $\beta_d(i, 1)$ is represented as $\beta_d(i)$. Ex_i is defined as the soft output of the block D_i , also known as extrinsic output, and is calculated using α_i and β_i . The processing time of a block, represented by a time-slot in Fig. 3, is the maximum time required to process α_i , $\beta_d(i, k)$, β_i and Ex_i or to store D_i .

α Memory: After the dummy β metric calculations on q data blocks are finished, the actual β metric calculation on p data blocks starts. β is calculated first in the p th data block, then in the $(p-1)$ th data block and so on. α is calculated in an order opposite to that of β . Since, the calculation of the extrinsic information Ex_i requires both α_i and β_i , all the α metrics for the p data blocks needs to be stored in p sub-banks. One additional sub-bank is needed to store the newly calculated α metrics since it is calculated continuously. Therefore, a total of $U_\alpha = (p+1)$ sub-banks are required for α metric calculation and storage.

Data Memory: For the sake of analysis, we consider p data blocks as a group. Then, the starting coordinate of dummy β calculation for the n th group is $P_n = (np-q, np+q)$, where $n = 1, 2, \dots, \lfloor K/p \rfloor$. Thus, $(np-q)$ is the time-slot at which the dummy calculations starts, and during this time-slot, data block $(np+q)$ is stored into a data memory sub-bank in reverse order. Note that dummy calculations are done directly on this data. Also, data block $(np+q-1)$ should have already been stored into a data memory sub-bank.

Let O_{xm} be the time interval for the m th group such that the data in one or more sub-banks are obsolete. The coordinates at which β calculation begin, i.e., the end of α metric calculation and dummy β calculation, is (mp, mp) . Once the β calculation of one data block finishes, the extrinsic information for this data block also finishes, and the corresponding sub-bank contains obsolete data. Thus this sub-bank is available for new data at time-slot $mp+1$. Thus O_{xm} lies between time-slot $mp+1$ and $(m+1)p$, i.e., $O_{xm} \in [mp+1, (m+1)p]$, where $m = 1, 2, \dots, \lfloor K/p \rfloor$.

By analyzing the relationship between the sub-banks that contain obsolete data and the starting point of dummy calculation, we can determine the number of sub-banks that are needed to store data. For any m , we can find a P_n that falls inside the time interval O_{xm} . Therefore, P_n satisfies the following constraint: $mp+1 \leq np-q \leq (m+1)p$. Since n is an integer, we have $n = \lceil m + (q+1)/p \rceil$. The number of time-slots, B , when the data becomes obsolete before the dummy β

TABLE I
OPTIMAL MEMORY LAYOUT FOR $\eta = 2$, $M = 1$

Time Slot	Data Memory				α Memory			Dummy	Output
	0	1	2	3	0	1	2		
0	$-D_1+D_2$.	.	.	$+\alpha_1$
1	D_1	$-D_2+D_3$.	.	α_1	$+\alpha_2$.	$\beta_d(2)$.
2	D_1	$-D_2$	$-D_3+D_4$.	α_1	$-\alpha_2$	$+\alpha_3$.	β_2, Ex_2
3	$-D_1+D_5$	D_3	$-D_4$.	$-\alpha_1$	$+\alpha_4$	α_3	$\beta_d(4)$	β_1, Ex_1
4	$+D_6$	$-D_5$	D_3	$-D_4$	$+\alpha_5$	$-\alpha_4$	α_3	.	β_4, Ex_4
5	$-D_6$	D_5	$-D_3$	$+D_7$	α_5	$+\alpha_6$	$-\alpha_3$	$\beta_d(6)$	β_3, Ex_3
6	$-D_6$	D_5	$+D_8$	$-D_7$	α_5	$-\alpha_6$	$+\alpha_7$.	β_6, Ex_6

starts for P_n is given by $B = (np-q) - (mp+1)$. Since data in one sub-bank is processed per time-slot, the number of sub-banks that contain the obsolete data is the same as the number of time-slots B .

In order to calculate the minimum number of sub-banks that are needed to store data, we need to calculate the amount of "live" data. Consider processing of data blocks $mp+1$ to $(m+1)p$. At this time, calculations on data block $(m-1)p$ is completed. To start the processing of dummy β at P_n we need to store data in the sub-banks up to $np+q-1$. Therefore, the total number of sub-banks to be stored in the memory is $(np+q-1) - (m-1)p$.

Since B sub-banks contain obsolete data, the minimum number of sub-banks that are needed to store input data is

$$U_{\text{data}} = [(np+q-1) - (m-1)p] - B = p+2q. \quad (3)$$

Therefore, the total number of sub-banks is

$$U_{\text{total}} = U_\alpha + U_{\text{data}} = 2(p+q) + 1. \quad (4)$$

The optimal memory configuration for η therefore consists of $2(p+q) + 1$ sub-banks: $p+1$ sub-banks are used for α metrics and $p+2q$ sub-banks are used to store input data. Fig. 4(a) and (b) describes the size of each sub-bank and the number of sub-banks as a function of η . Since the sub-bank size is a function of L/q , the sub-bank size is the same for all η that have the same q (such as $\eta = 1/2, 3/2, \dots$).

Total Memory: The total memory requirement for different η values is derived as follows. Each word in α memory is of size W_α and contains the α values for all the states in one stage or the trellis. Similarly, each word in data memory is of size W_{data} and contains the branch metrics for all the states in one stage of trellis. The total memory for the optimal configuration is the sum of the α memory and the data memory ($L/q [(p+2q) \cdot W_{\text{data}} + (p+1) \cdot W_\alpha]$).

Optimal values of W_{data} and W_α have been derived in [9] based on BER requirements. For the sake of simplicity, we assume that each stage of α memory is of size W and each stage of data memory is of size $2W$. Therefore, the total memory can be defined as

$$\text{Total memory} = \frac{(3p+4q+1)WL}{q}. \quad (5)$$

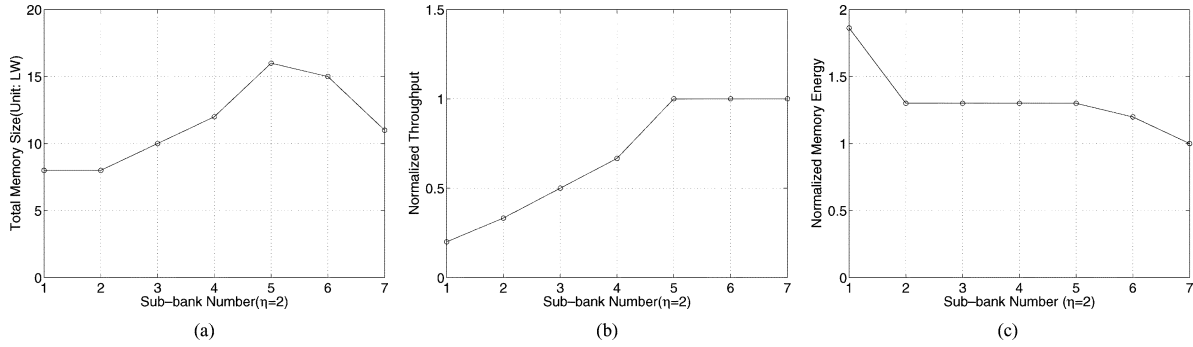


Fig. 5. Results for the nonoptimal sub-banked configuration as a function of number of sub-banks for $\eta = 2$, $M = 1$. (a) Total memory size. (b) Normalized throughput. (c) Normalized memory energy.

Note that the size of the memory can be reduced if some of the stages are not stored and data is recomputed on-the-fly from raw input [2].

Fig. 4(c) describes the variation in the total memory size as a function of η . While the memory size increases with increase in η , the number of memory accesses is the same for $\eta \geq 1$. This is because the memory accesses are needed only for the forward and backward recursion calculations (the dummy calculations do not need any access; they can be done in parallel with data memory writes).

The variation in memory energy for $\eta \geq 1$ is likely to be very small. Assuming that the memory access energy increases by 30% as the memory size increase by a factor of two, the variation in access energy of the sub-banks is likely to be compensated by the overhead due to use of multiple sub-banks. The variation in computation energy is a lot more significant. As η increases, the total number of dummy calculation ($= N/\eta$) decreases, and the computation energy decreases.

Example: Optimal single-port sub-banked structure for $\eta = 2$

Table I shows the single-port sub-bank structure for $\eta = 2$, $M = 1$ [Fig. 2(b)]. The optimal configuration consists of seven memory sub-banks: four sub-banks are needed for storing input data (shown under data memory column) and three sub-banks are needed for α metrics (shown under α memory column). No sub-bank is required for storing $\beta_d(i)$ because only the last dummy metric value is needed. There is also no need to store β_i as it is immediately used up for the calculation of $E x_i$. As soon as $E x_i$ for data block i is generated, the corresponding data and α values are obsolete and are replaced by the new values. A “+” sign in Table I shows a write pointer while a “-” sign shows a read pointer, those sub-banks without a “+” or “-” sign remain unchanged during the time slot. The total number of α and β calculations is the same as the data frame size. Dummy- β calculations are done in every other time-slot.

B. Optimal Dual-Port Sub-Banking for $\eta = p/q$, $M = 1$

Unlike the single-port memory where we need an extra sub-bank to store the newly calculated α metrics or input data, in the dual-port memory, the newly calculated α metric or input data can be written into the sub-bank which contains the obsolete α metric or data. The dual access capability results in one less memory sub-bank for both α memory and data memory. Therefore, the total number of dual-port sub-banks is $2(p+q) - 1$. The optimal dual-port sub-banking structure requires total memory of $(3p + 4q - 2)WL/q$.

Fig. 4 also describes the sub-bank size, number of sub-banks and total memory size as a function of η for dual-port memories. While the total memory size in LW units is lower for dual-port memories, the area of each cell in a dual-port memory is larger, making the actual silicon area of dual-port memory larger. In fact, analysis of the throughput, area, and power consumption of single-port and dual-port structures showed that the single-port sub-banking structure performs

better than the dual-port sub-banking structure in terms of throughput, power consumption, and area.

C. Effect of Nonoptimal Memory Configuration

If the number of sub-banks is smaller than the optimal, the throughput, total memory size and memory energy are affected. This is illustrated in Fig. 5 for the case of $\eta = 2$. The throughput has been normalized with respect to the throughput of the optimal configuration for $\eta = 2$. If the number of sub-banks is reduced from 7 (optimal) to 5, the throughput can be maintained only if the size of each sub-bank is increased significantly. This causes a significant increase in memory. If the number of sub-banks is reduced to (say) 3, then the total memory size is not affected as much but the throughput drops by a factor of 2. Thus there is a large price to be paid if the number of sub-banks is less than the optimal for a given η .

D. Tradeoffs

In the memory design space exploration, we have considered three optimization parameters: throughput, memory size and energy. As η increases, the throughput reduces slightly, the memory size increases [Fig. 4(c)] and the overall energy reduces (since computation energy reduces and memory energy increases mildly). Thus choice of η depends on the relative importance of these parameters. We ignore the effect of coding performance on η since BER decreases very mildly with increase in η .

For a fixed η , the proposed optimal sub-banked memory configuration achieves the highest throughput, lowest memory size and lowest energy. However, if there are additional architectural constraints, such as memory size or number of sub-banks, then analysis such as the one presented in Fig. 5 will help in choosing the right configuration.

E. Comparison With Previous Work

Multiple sub-bank memory organizations of APP decoder were proposed in [4] and [5]. The SW structure used in [4], [5] is actually the case $\eta = 1$ in this paper. While our optimal sub-banking structure for $\eta = 1$ requires three single port memory sub-banks to store input data and two sub-banks to store α memory, the configuration in [4] requires three sub-banks to store input data and only one sub-bank to store computed α . While the number of sub-banks in [4] is smaller, the number of simultaneous operations is larger. In each time-slot, new data is loaded into a sub-bank, and α , dummy- β , β metrics and extrinsic information are calculated. The reduction in number of sub-banks to store α is achieved by using a unique type of RAM with read-modify-write access in a single cycle. If instead one additional α memory bank is introduced, the writing of new data or α value becomes independent of the active read operations, and the access for each stage can be reduced to one per memory sub-bank as demonstrated in this paper. [5]

uses dual-port memories to reduce the number of RAMs to three: two for input data and one for α metrics. This is the same as our dual port configuration for $\eta = 1$. The work presented in [3] describes the variations in area, throughput and memory energy for monolithic memories for different values of η . While the throughput variations are very similar to those presented here, the memory energy is significantly different – sharp increase in [3] vs. almost constant energy with increase in η in our architecture. This difference could be due to differences in data access pattern, use of monolithic memory, etc.

IV. CONCLUSION

In this paper, the optimal memory sub-bank structure for a MAP-based SISO decoder that achieves a very high throughput for different SW configuration was derived. A tradeoff between performance parameters such as throughput/decoding latency, and architectural parameters such as number of sub-banks, memory size, and memory energy was established.

The SW approach reduced the latency of the SISO decoder and thereby reduced the overall decoding latency of the iterative Turbo decoder. For very high throughput Turbo decoders, the interleaver/de-interleaver memory access is clearly a bottleneck. Interleaving techniques such as those proposed in [10] in conjunction with the proposed SISO decoder architecture can help realize ultra high throughput Turbo decoders.

REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in *Proc. Int. Conf. Commun.*, 1993, pp. 1064–1070.
- [2] Z. Wang, Z. Chi, and K. K. Parhi, "Area-efficient high-speed decoding schemes for Turbo decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 6, pp. 902–912, Dec. 2002.
- [3] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of MAP Turbo decoder algorithms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 2, pp. 305–312, Apr. 2001.
- [4] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architecture for Turbo codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 3, pp. 369–379, Sep. 1999.
- [5] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 3, pp. 279–285, Jun. 2002.
- [6] M. M. Mansour and N. R. Shanbhag, "VLSI architecture for SISO-APP decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 4, pp. 627–650, Aug. 2003.
- [7] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.
- [8] J. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structure for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1661–1671, Feb./Mar./Apr. 1994.
- [9] Z. Wang, H. Suzuki, and K. Parhi, "Finite wordlength analysis and adaptivdecoding for Turbo/MAP decoders," *J. Very Large Scale Integr. (VLSI) Syst. Signal Process.*, vol. 29, pp. 209–221, 2001.
- [10] Z. Wang and K. K. Parhi, "High performance, high throughput Turbo/SOVA decoder design," *IEEE Trans. Commun.*, vol. 51, no. 4, pp. 570–579, Apr. 2003.

Instruction Code Mapping for Performance Increase and Energy Reduction in Embedded Computer Systems

Sri Parameswaran and Jörg Henkel

Abstract—In this paper, we present a novel and fast constructive technique that relocates the instruction code in such a manner into the main memory that the cache is utilized more efficiently. The technique is applied as a preprocessing step, i.e., before the code is executed. Our technique is applicable in embedded systems where the number and characteristics of tasks running on the system is known *a priori*. The technique does not impose any computational overhead to the system. As a result of applying our technique to a variety of real-world applications we observed through simulation a significant drop of cache misses. Furthermore, the energy consumption of the whole system (CPU, caches, buses, main memory) is reduced by up to 65%. These benefits could be achieved by a slightly increased main memory size of about 13% on average.

Index Terms—Cache memories, energy conservation, low power.

I. INTRODUCTION

In embedded systems containing a memory hierarchy, the performance can often be improved by reducing the overhead of instruction cache misses. Along with that often comes also a reduction in energy consumption of the whole system: assumed, a basic system consists at least of a CPU, an instruction cache, a main memory and a bus system, then the following mechanisms can reduce the power/energy consumption of the whole system:

- lesser cache misses lead to lesser main memory accesses and thus to a reduced energy consumption of the main memory;
- reduced cache misses decreases the traffic on the bus and thus the energy/power consumption on the buses;
- lesser waiting cycles will be imposed to the CPU and such reducing the CPU energy/power consumption.

The potential savings in energy vary by system. As an example, Liu *et al.* have estimated for memories that the total power savings can be up to 70% of the total power [4].

Cache misses can be classified as follows: *compulsory misses*, *conflict misses*, and *capacity misses*. A compulsory miss occurs when an instruction or data is referenced for the first time, a conflict miss occurs when data or instruction compete for the same cache location, and a capacity miss occurs when the size of the cache is too small for the amount of memory needed to execute the program under consideration. New processor architectures along with their application in mobile computing/communication/internet devices demand efficient cache architectures. To satisfy the fast access demands of the system, direct mapped caches are commonly used. While these satisfy the speed constraints, they increase the amount of conflict misses, since there is no alternate position for the requested data or instruction to occupy the cache (as is the case in set associative cache design). Modern processors contain both an instruction cache and a data cache. There have been several differing methods to reduce cache misses in both.

In our I-CoPES methodology, which is described in this paper, a novel scheme is introduced to reduce the before-mentioned instruction miss conflict. Our methodology uses a constructive algorithm to

Manuscript received February 18, 2002; revised August 31, 2003.

S. Parameswaran is with the School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia (e-mail: sridevan@cse.unsw.edu.au).

J. Henkel is with NEC Laboratories America, Princeton, NJ 08540 USA (e-mail: henkel@nec-lab.com).

Digital Object Identifier 10.1109/TVLSI.2004.842936