Searching for new convolutional codes using the cell broadband engine architecture

Johnsson, Daniel; Bjärkeson, Fredrik; Hell, Martin; Hug, Florian

Published in:
IEEE Communications Letters

DOI:
10.1109/LCOMM.2011.040111.101624

2011

Searching for New Convolutional Codes using the Cell Broadband Engine Architecture

Daniel Johnsson, Fredrik Bjärkeson, Martin Hell, Florian Hug

***Abstract*—The Bidirectional Efficient Algorithm for Searching code Trees (BEAST), which is an algorithm to efficiently determine the free distance and spectral components of convolutional encoders, is implemented for the Cell Broadband Engine Architecture, efficiently utilizing the underlying hardware.**

Exhaustive and random searches are carried out, presenting new rate $R = 1/2$ convolutional encoding matrices with memory $m = 26$–29 and larger free distances and/or fewer spectral components than previously known encoding matrices of same rate and complexity.

The main result of this paper consists in determining the previously unknown optimum free distance convolutional code with memory $m = 26$.

***Index Terms*—Convolutional codes, BEAST, Cell Broadband Engine Architecture**

I. Introduction

FINDING good convolutional codes by using algebraic methods has not been very successful, and commonly computer searches have provided the currently best known codes. Thus, algorithms for an exhaustive search of convolutional encoders remain an important research topic. Algorithms like the BEAST [1], [2]—Bidirectional Efficient Algorithm for Searching code Trees—provide a theoretical limit on the search-complexity, while their exact implementation is not specified.

In [3] approximately 80 IBM x86 Opteron cores have been used to conduct an exhaustive search for memory 25 convolutional codes. However, the increasing complexity of modern processors, with very fast execution of certain operations, provides new tools for efficient implementations. Utilizing extended instruction sets targeting specific processors, the practical efficiency of such algorithms can be improved. This paper focuses on the very cost-efficient Cell Broadband Engine [4], most notably used within the PlayStation 3™ gaming console, which has previously been efficiently used in several scientific applications.

This paper essentially follows [5]; in Section II basic principles of convolutional codes and their distance properties are introduced. The BEAST is described in Section III, while Section IV covers some of the specialities of the Cell Broadband Engine Architecture. The results obtained by exhaustive and random searches for rate $R = 1/2$ convolutional encoders of memory $m = 26$–29 using the BEAST on the Cell Broadband Engine Architecture are presented in Section VI, summarizing the contribution of this paper in Section VII.

Manuscript received September 2, 2010. The associate editor coordinating the review of this letter and approving it for publication was A. Burr.

The authors are with the Department of Electrical and Information Technology, Lund University, P.O. Box 188, SE-22100 Lund, Sweden (Email: {djohnsson, fredrik.bjarkeson}@gmail.com, {martin, florian}@eit.lth.se)

Digital Object Identifier 10.1109/LCOMM.2011.040111.101624

II. Preliminaries

Consider a rate $R = 1/2$ convolutional code $\mathcal{C}$ with *memory* m and *encoding matrix* [6, Ch. 2]

$$G(D) = \begin{pmatrix} \boldsymbol{g}_1(D) & \boldsymbol{g}_2(D) \end{pmatrix}$$

where $\boldsymbol{g}_i(D)$, $i = 1, 2$, denotes the ith binary generator polynomial of at most degree m.

We represent $G(D)$ by the semi-infinite encoding matrix $\boldsymbol{G}$ in the time domain; then the binary infinite information sequence $\boldsymbol{u}$ is mapped to the binary code sequence $\boldsymbol{v}$ by $\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G}$. While it is straight-forward to generalize these concepts to rate $R = b/c$ convolutional codes, we will for simplicity limit ourselves hereinafter to rate $R = 1/2$ convolutional codes with polynomial encoding matrices.

The *free distance* d_{free} of a convolutional code $\mathcal{C}$ is defined by

$$d_{\text{free}} = \min_{\boldsymbol{v},\boldsymbol{v}' \in \mathcal{C}, \boldsymbol{v} \neq \boldsymbol{v}'} \left\{ d_{\text{H}}\left(\boldsymbol{v}, \boldsymbol{v}'\right) \right\} = \min_{\boldsymbol{v} \in \mathcal{C}, \boldsymbol{v} \neq \boldsymbol{0}} \left\{ w_{\text{H}}\left(\boldsymbol{v}\right) \right\}$$

where $d_{\text{H}}(s, s')$ and $w_{\text{H}}(s)$ denote the *Hamming distance* and the *Hamming weight*, respectively [6, Sec. 1.2]. The free distance and the *free distance spectrum* [7] determine the error-correcting capabilities of a convolutional encoder.

III. The BEAST

The BEAST [1], [2]—Bidirectional Efficient Algorithm for Searching code Trees—is an efficient algorithm used for finding the spectral components of block codes and convolutional encoders as well as for maximum-likelihood decoding of block codes.

Consider a *code tree* of a rate $R = 1/2$, memory m convolutional encoder, in which every *node* ξ represents one of the 2^m different states $\boldsymbol{\sigma}(\xi)$. Each of the two branches emerging from every node at depth i are labeled by an input bit u_i and an output tuple $v_i^{(1)} v_i^{(2)}$. Moreover, for every node ξ denote its unique parent node by ξ^{P} and its two children nodes by ξ^{C}.

Clearly, every codeword $\boldsymbol{v}$ of a noncatastrophic convolutional generator matrix [6, Sec. 2.1] corresponds to a path through such a code tree, $\xi_{\text{root}} \rightarrow \xi_{\text{toor}}$, with $\boldsymbol{\sigma}(\xi_{\text{root}}) = \boldsymbol{\sigma}(\xi_{\text{toor}}) = \boldsymbol{0}$.

Assume searching for the number of codewords n_w, that is, the number of paths $\xi_{\text{root}} \rightarrow \xi_{\text{toor}}$ of Hamming weight w. For each such path there exists an intermediate node ξ, such that

$$w_{\text{F}}(\xi) = f_w + j \quad w_{\text{B}}(\xi) = b_w - j \quad j = 0, 1$$

where w_{F} and w_{B} denote the accumulated Hamming weights starting at the root and toor node, respectively, and f_w, b_w are freely chosen integers such that $f_w + b_w = w$. Note, although

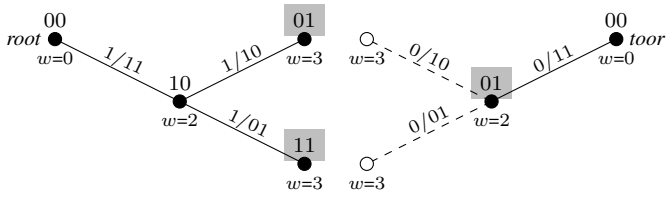

Fig. 1. Forward and Backward code tree explored by the BEAST with forward weight $f_w = 3$ and backward weight $b_w = 2$ with encoding matrix $G(D) = (1 + D + D^2 \;\; 1 + D^2)$.

f_w and b_w may be chosen freely, an uneven weight distribution decreases the efficiency of the BEAST.

Based on these observations, the BEAST finds the number of codewords n_w of Hamming weight w by conducting a bidirectional search as follows:

1) *Forward Search*: Starting from the zero-weight root node ξ_{root}, build up a forward code tree and obtain the set of nodes $\mathcal{F}_{+j}$, $j = 0, 1$, satisfying
$$\mathcal{F}_{+j} = \left\{ \xi \middle| w_{\text{F}}(\xi) = f_w + j,\; w_{\text{F}}(\xi^{\text{P}}) < f_w,\; \boldsymbol{\sigma}(\xi) \neq \mathbf{0} \right\}.$$
2) *Backward Search*: Starting from the zero-weight toor node ξ_{toor}, build up a backward code tree and obtain the set of nodes $\mathcal{B}_{-j}$, $j = 0, 1$, satisfying
$$\mathcal{B}_{-j} = \left\{ \xi \middle| w_{\text{B}}(\xi) = b_w - j,\; w_{\text{B}}(\xi^{\text{C}}) > b_w,\; \boldsymbol{\sigma}(\xi) \neq \mathbf{0} \right\}.$$
3) *Match*: Determine the number of codewords n_w of Hamming weight w by finding the number of node pairs $(\xi, \xi') \in \mathcal{F}_{+j} \times \mathcal{B}_{-j}$, $j = 0, 1$, with the same state,
$$n_w = \sum_{j=0}^{c-1} \sum_{(\xi,\xi') \in \mathcal{F}_{+j} \times \mathcal{B}_{-j}} \chi(\xi,\, \xi') \tag{1}$$
$$\chi(\xi,\, \xi') = \begin{cases} 1, & \text{if } \boldsymbol{\sigma}(\xi) = \boldsymbol{\sigma}(\xi') \\ 0, & \text{otherwise.} \end{cases}$$

Example 1 *Consider the encoding matrix $G(D) = (1 + D + D^2 \;\; 1 + D^2)$ and assume we want to determine the number of codewords of weight 5. Applying the BEAST with, for example, forward weight $f_w = 3$ and backward weight $b_w = 2$, leads to the two code trees illustrated in Fig. 1.*

According to the algorithm, the forward set stores all nodes with weight 3 and 4 (marked in gray), that is, $\mathcal{F}_{+0} = 01, 11$ and $\mathcal{F}_{+1} = \emptyset$, while the backward set contains nodes with weight 1 and 2 (marked in gray), that is, $\mathcal{B}_{-0} = 01$ and $\mathcal{B}_{-1} = \emptyset$. Matching the corresponding sets, we obtain one common node 01 and thus conclude that there exists one codeword of weight 5.

IV. The Cell Broadband Engine Architecture

The Cell Broadband Engine Architecture (CBEA) is a heterogeneous processor architecture, originally developed for the PlayStation 3™ by Sony, Toshiba and IBM. The processor in the PlayStation 3™ is equipped with one general-purpose *PowerPC Processor Unit* (PPU) and seven[1] *Synergistic Processor Units* (SPU) [4].

[1]A GNU/Linux operating system is run on the PPU within a hypervisor, supervised by the native GameOS, permitting access to only six SPUs.

IBM distributes a Cell software development kit (SDK) [4] for the PlayStation 3™ containing tools for finding performance bottlenecks and bugs. An included system simulator can be used for visualizing a near instruction accurate simulation of a Cell processor. The SDK also contains an assembly visualizer for aiding in manually ordering instructions in the dual SPU pipeline in order to reduce the amount of stalling, as well as a feedback directed program restructuring tool for automating the same task.

Each SPU is equipped with a Memory Flow Controller (MFC), capable of transferring data asynchronously without interrupting program execution, as well as a 256 KB software-controlled SRAM-based Local Store (LS) containing both data and instructions. Utilizing direct memory access commands, data can be copied between the main memory and the LS on 16-byte boundaries, with up to 16 simultaneous transfers in flight. Consequently, using the MFC to asynchronously transfer data between the SPU and PPU, the communication delay between each SPU and its corresponding thread on the PPU can be largely reduced. Moreover, as the SPUs lack data, instruction and branch caches, fine grained control of the processor can be used for further individual optimizations.

The PPU and SPU-units have instruction set architectures (ISA) operating on 128-bit data types which allow simultaneous processing of four separate code tree branches [5]. For example, with the SPU ISA containing an instruction for counting the number of active bits in a byte (`cntb`), the Hamming weight calculations of four 32-bit `int`s can be effectively calculated simultaneously using the C-function:

```
inline vec_uint4 hw(const vec_uint4 v) {
    return (vec_uint4) spu_sumb(
            spu_cntb((vec_uchar16) v),
            spu_splats((uint8_t) 0)); }
```

Conducting an exhaustive code search involves processing large amounts of data-independent generator matrices. Using these ideal data characteristics, every SPU processes a generator matrix independently using data-parallelism, in order to take full advantages of all CBEA processor cores.

V. Implementation

As the row distances upper-bounds the free distance [6, Sec. 3.1], we start by using the so-called *row distance test* [6, Sec. 8.2] [8] to remove nonpromising encoders without losing optimality [5, Sec. 6.1]. After checking the remaining encoders to be noncatastrophic [6, Sec. 2.1], the BEAST is used to determined the remaining encoder properties, like the free distance d_{free} and the spectral components. In particular, as the row distance test extensively uses the previously defined Hamming weight-function, it can be implemented very efficiently on the SPUs.

Analyzing pre-generated sets with empirical methods, a near-optimal time-tradeoff between the BEAST and the execution time of the rejection algorithm can be achieved. For example, in case of the exhaustive search carried out for memory $m = 26$, the overall amount of approximately $1.68 \cdot 10^{15}$ encoders could be reduced by a factor of 10^6, still taking up to 12 GB storage space with each encoder being stored as two 32-bit polynomials.

TABLE I
NEWLY FOUND GENERATOR POLYNOMIALS FOR MEMORIES 26–29 OBTAINED BY EXHAUSTIVE (OFD) AND RANDOM (RND) SEARCHES, COMPARED TO PREVIOUSLY KNOW BEST ODP ENCODING MATRICES.

m	$g_1(D)$	$g_2(D)$	d_{free}	Spectrum	Note
26	645055711_8	525626523_8	28	$24, 58, \ldots$	ODP [3]
	736110763_8	426237051_8	28	$9, 66, \ldots$	OFD
27	7270510714_8	5002176664_8	28	$1, 28, \ldots$	ODP [3]
	6276631214_8	5475602164_8	29	$19, 63, \ldots$	RND
28	7605117332_8	5743521516_8	30	$54, 0, \ldots$	ODP [3]
	6005305632_8	5762423076_8	30	$53, 0, \ldots$	RND
29	7306324763_8	5136046755_8	30	$5, 47, \ldots$	ODP [3]
	6026566375_8	5713575517_8	31	$64, 164, \ldots$	RND

The BEAST can be efficiently implemented using a recursive depth-first method by partitioning the problem set and using the SPUs executing in parallel to calculate and sort subsets of the forward and backward sets for a given w. Calculated subsets are transferred asynchronously from the SPU LS to main memory using the MFC while each SPU continues to produce the next subset. Once both complete sets have been produced, the PPU is used to find a common state (1) yielding the d_{free} and if none is found, the process is repeated using a greater w.

However, using the recursive method and encoding matrices with increasing memory sizes, a large call stack memory size is needed. As the size of the SPU LS is limited, the use of an iterative implementation greatly reduces the memory footprint of the algorithm. In particular, eliminating the successive recursive calls and replacing the call stack with a bit stack, it is possible to reduce the memory overhead for each depth in the code tree from 160 bytes to 10 bits [5, Sec. 5.3].

Instead of storing the state information at every depth and relying on the call stack to restore the previous state and its weight, the iterative implementation keeps track of the information lost in a state transition and re-calculates the previous state and its weight when needed. However, due to the additional calculations, the efficiency of the BEAST is reduced slightly.

VI. SEARCH RESULTS

Using the implementation as discussed above, an exhaustive search for rate $R = 1/2$ convolutional encoders with memory $m = 26$ has been carried out, resulting in the previously unknown *optimum free distance* (OFD) encoding matrix being presented in Table I in octal notation with zeros padded from the right, *i.e.,* $46_8 = 100\ 110_2 = 1 + D^3 + D^4$.

With increasing memory, an exhaustive search becomes infeasible and searches are performed either randomly or are limited to small subsets of convolutional encoders with certain "good enough" properties [8]. For example, in [3], a search limited to *optimum distance profile* (ODP) encoding matrices was performed, leading to near-optimum encoding matrices.

By running a random search for rate $R = 1/2$ encoding matrices with memory $m = 27 - 29$ on the Cell Broadband Engine Architecture, encoders with better properties, that is, larger free distances and/or fewer spectral components could be found. These newly obtained encoders in comparison to the previously found best ODP encoders are additionally given in Table I.

With the free distance for the OFD convolutional codes with memory $m = 25$ and $m = 26$ being the same, the complexity of the exhaustive search increases roughly with a factor of four. Compared to the exhaustive search with memory $m = 25$ with approximately 80 IBM x86 Opteron cores with 2.6 Ghz and 4 GB memory in [3], only five PlayStation 3™ were used to conduct the corresponding search for memory $m = 26$. Nevertheless, the overall running time remained the same, namely, approximately two months in both cases.

VII. CONCLUSIONS

The BEAST has been implemented on the Cell Broadband Engine Architecture, focusing on efficiently exploiting the underlying heterogeneous system architecture. Potential bottlenecks have been highlighted and ways to achieve an efficient implementation have been provided.

New rate $R = 1/2$ convolutional encoders with memory $m = 26$–29 with better free distances and/or better distance spectra than previously known ODP encoders of same rate and complexity have been presented. For memory $m = 26$, an exhaustive search could be conducted, leading to our main result, the previously unknown OFD encoding matrix.

As the PlayStation 3™ was introduced in 2006, using newer processor architectures like GPUs or homogeneous multicore CPUs might lead to even more efficient implementations. Such implementations might be used to find the still unknown OFD convolutional codes with slightly larger memories.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] I. E. Bocharova, M. Handlery, R. Johannesson, and B. D. Kudryashov, "A BEAST for prowling in trees," in *Proc. 39th Annual Allerton Conf. Commun., Control, and Computing*, Monticello, Illinois, USA, Oct. 2001.
[2] ——, "A BEAST for prowling in trees," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 1295–1302, Jun. 2004.
[3] F. Hug, "On graph-based convolutional codes," Master's thesis, Lund Institute of Technology, Lund, Sweden, 2008.
[4] (2009, Dec.) Cell broadband engine resource center. [Online]. Available: http://www.ibm.com/developerworks/power/cell/
[5] D. Johnsson and F. Bjärkeson, "Playing with the BEAST," Master's thesis, Lund Institute of Technology, Lund, Sweden, 2009. [Online]. Available: http://www.eit.lth.se/researchprojects/141/thesis-cell.pdf
[6] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. Piscataway, NJ: IEEE Press, 1999.
[7] I. E. Bocharova, F. Hug, R. Johannesson, and B. D. Kudryashov, "A note on convolutional codes: Equivalences, MacWilliams identity, and more," Jun. 2009, submitted to IEEE Trans. on Inf. Theory.
[8] R. Johannesson, "Robustly optimal rate one-half binary convolutional codes," *IEEE Trans. Inf. Theory*, vol. 21, no. 4, pp. 464–468, Jul. 1975.