Data Compression via Logic Synthesis

Luca Amarú[1], Pierre-Emmanuel Gaillardon[1], Andreas Burg[2] and Giovanni De Micheli[1]

[1]Integrated Systems Laboratory (LSI), EPFL, Switzerland
[2]Telecommunication Circuits Laboratory (TCL), EPFL, Switzerland

***Abstract*—Nowadays, most software and hardware applications are committed to reduce the footprint and resource usage of data. In this general context, lossless data compression is a beneficial technique that encodes information using fewer (or at most equal number of) bits as compared to the original representation. A traditional compression flow consists of two phases: data decorrelation and entropy encoding. Data decorrelation, also called entropy reduction, aims at reducing the autocorrelation of the input data stream to be compressed in order to enhance the efficiency of entropy encoding. Entropy encoding reduces the size of the previously decorrelated data by using techniques such as Huffman coding, arithmetic coding, and others. When the data decorrelation is optimal, entropy encoding produces the strongest lossless compression possible. While efficient solutions for entropy encoding exist, data decorrelation is still a challenging problem limiting ultimate lossless compression opportunities. In this paper, we use logic synthesis to remove redundancy in binary data aiming to unlock the full potential of lossless compression. Embedded in a complete lossless compression flow, our logic synthesis based methodology is capable to identify the underlying function correlating a data set. Experimental results on data sets deriving from different causal processes show that the proposed approach achieves the highest compression ratio compared to state-of-art compression tools such as ZIP, bzip2 and 7zip.**

I. Introduction

The maturity of *Electronic Design Automation* (EDA) tools enables today's billions transistors chip to be designed and tested efficiently. Such great success drives the application of EDA algorithms to non-traditional EDA fields, e.g., cure of cancer [1], smart water [2], cure of genetic diseases [3], smart grid [4], etc. In this work, we use EDA synthesis techniques to reduce the footprint and resource usage of binary data.

It is estimated that the total amount of data stored in world's digital devices could be compressed by a factor of 4.5x without any loss of information [5]. The efficiency of data compression techniques is key to securing a profitable usage of physical resources. For this reason, data compression is a widely studied and active research field. Original compression methods, such as Huffman and arithmetic coding, are based on the Shannon entropy [6]. The Shannon entropy quantifies the information contained in data. When the considered data is a sequence of *independent and identical distributed* (i.i.d.) *Random Variables* (RVs), Shannon entropy provides the upper bound on the best possible lossless compression. In this condition, Huffman and arithmetic coding methods are optimal since they are proven to asymptotically achieve the Shannon entropy. Unfortunately, most of the real-world information cannot be modeled as sequence of i.i.d. RVs. Indeed, digital data is often strongly correlated. In order to overcome this limitation, succeeding techniques have been developed embedding a decorrelation pre-processing step prior (or integrated) to entropy encoding based compression [6]. Notably, dictionary techniques incorporate the structure of correlated data, by building a list of frequently occurring patterns, to increase the compression ratio. This compression approach has been used in microprocessor architectures, e.g., Thumb from Arm, where the instruction stream is compressed in I-memory and decompressed in the processor [7]. The *Lempel-Ziv* (LZ) method [8], and its evolutions, represents a very successful dictionary based compression technique employed in most practical compression standards. However, LZ-based methods imply some assumptions on the pattern recurrence locality. Moreover with dynamic dictionaries the efficiency of the compression converges slowly with the size of the data. For this reason, specialized decorrelation transforms have been successively studied with the aim at further improving the compression ratio [6]. Among existing decorrelation transforms, fixed-basis (heuristic) transforms, such as discrete cosine transform, are used in practice while variable-basis (optimal) transforms, such as KLT [6], are used to bound the best theoretical transform performance. Indeed, the variability of the transform basis is not a desirable property in data compression. Fixed-basis transforms usually target a specific class of data, for example the discrete cosine transform is advantageous for imaging (JPEG standard [10]).

A major aim for today's compression tools is to overcome the limits of traditional decorrelations techniques in order to unlock the full potential of lossless data compression.

In this paper, we use logic synthesis to compact the size of causal data sets enabling novel lossless compression opportunities. Logic synthesis is the process by which an abstract Boolean function description is transformed into a corresponding minimized logic circuit [11]. In the data compression context, the capability of modern logic synthesis tools to identify and remove redundancy, while preserving the initial behavior of the circuit, paves the way for a general lossless compression approach with the aim to find the underlying logic function generating the data we want to compress. Experimental results on data sets derived from causal processes show that our compression method based on logic synthesis reduces the data size by 1∼2 orders of magnitudes compared to state-of-art compression tools such as ZIP, bzip2 and 7zip.

The remainder of this paper is organized as follows. Section II first provides background on lossless data compression and logic synthesis, then it discusses the contributions of this work compared to prior art on compression techniques performed via Boolean minimization. In Section III, the proposed logic synthesis based compression methodology is presented. Experimental results for our compression approach are presented and compared with state-of-art compression standards in Section IV. We conclude the paper in Section V.

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

II. Background and Motivation

This section first presents background on data compression and logic synthesis. Later, it discusses the contributions of this work compared to prior art on compression techniques performed via Boolean minimization. Please note that, in this work, we focus only on lossless data compression.

A. Data Compression

A *lossless* compression scheme refers actually to two algorithms: (i) a compression algorithm that takes in input a data sequence X and reduces it to X_c (that requires fewer, or at most equal, bits than X), and (ii) a reconstruction algorithm that recovers *exactly* X from X_c, i.e., with no loss of information. In other words, in a lossless compression scheme the compression algorithm must be *reversible* [6]. Given the dual nature of *lossless* compression schemes, we concentrate hereafter only on compression algorithms provided that they satisfy the reversibility requirement.

Nowadays, practical compression approaches are based on two subsequent phases: first, the input data is decorrelated and then entropy encoding techniques are applied as final compression step.

1) Data Decorrelation (Entropy reduction): Data decorrelation, also called entropy reduction, is a data preprocessing phase aiming at reducing the autocorrelation of the input data.

Linear transformations are efficient means to accomplish this task [6]. Among all the decorrelation transforms, the *best* coding gain, in term of compression ratio, is provided by the *Karhunen-Loeve Transform* (KLT). The KLT is a transform having signal-dependent basis and random variables as coefficients. In the discrete (binary) domain, the KLT is described by a matrix having as columns the eigenvectors of the autocorrelation matrix of the input (binary) sequence considered [6]. The signal-dependency of the KLT makes its use inefficient in practical applications where the basis (or the data autocorrelation itself) must be provided to the decoding algorithm which is unaware of the properties of the compressed data. Such overhead removes the advantage of the KLT theoretical optimality. For this reason, the KLT is typically used as theoretical bound for the best coding gain achievable by decorrelation transforms. On the other hand, practical decorrelation techniques of interest make assumptions on the original data properties resulting in fixed basis transforms. A well-known example is the *Discrete Cosine Transform* (DCT), widely used in image compression methods, e.g., JPEG [10], which employs cosine functions as fixed transform basis. Other specialized transforms have been developed in literature, e.g., BCJ/BCJ-2 [20] for binary executables, Burrows-Wheeler [9] especially efficient in the field of bio-informatics etc.

Another notable approach to decorrelate data is based on dictionary techniques. The core idea of dictionary techniques is to build a dictionary (static or dynamic) of recurring patterns in the input data. Such patterns are directly encoded by their indexes in the dictionary. The *Lempel-Ziv* (LZ) method [8] is one of the most recognized dictionary based compression technique used in many practical compression standards. The general efficiency of existing dictionary techniques is limited by (i) assumptions on pattern recurrence locality and (ii) issues related to dictionary flexibility.

We refer the interested reader to [6] for an extended and complete discussion of data decorrelation.

2) Entropy Encoding: Entropy encoding refers to a class of (lossless) coding techniques able to compress an input data down to its entropy. When the entropy information is defined according to the exact probabilistic model, entropy encoding achieves the optimum compression for any input data. However, the correlation and the (often complex) underlying function that produced the data set is typically not known. Hence, the choice of the right probabilistic model is a difficult problem usually simplified by decorrelating transforms. Once the data is (fully) decorrelated, a simple stochastic model is reliably employable to define the information entropy. Then, entropy encoding methods can be applied successfully. Original entropy encoding techniques are Huffman coding [12] and arithmetic coding [6] that form the basis of current compression software and standards. For a review of such methods, we refer again to [6].

B. Logic Synthesis

Logic synthesis is the process by which (virtually) all digital integrated circuits are designed [11]. Logic synthesis aims to transform a general description of a Boolean function into its minimal logic circuit implementation. The development of synthesis algorithms have been driven by the exponential growth of digital electronics, requiring contemporary tools to be scalable, efficient and capable to produce near-optimal results. Logic synthesis shares optimization criteria with many other problems, consequently its application to non-traditional (non-EDA) fields is attracting the interest of researchers in several science communities.

For the sake of brevity, we do not review basic concepts and notation for logic synthesis, please see [11] for a review.

C. Contribution to Prior Art

Previous works in [13]–[15] explored the possibility to compress data using Boolean minimization. Their method consist in treating a binary sequence of 2^N elements as a complete truth table for a N-input single output Boolean function. Two-level minimization is applied in [13]–[15] to lossless compress the information stored in such truth table (representing the initial sequence). We differentiate from these methods by (i) the use of a more sophisticated initial SOP representation in place of exhaustive truth tables, (ii) extension from single output to multioutput function representation to enhance the manipulation flexibility, (iii) introduction of encoding/decoding *arrow of time* in binary strings to facilitate the logic synthesis task and (iv) identification and selective manipulation of uncorrelated data with traditional entropy encoding methods.

III. Logic Model for Data Compression

Binary data is generated, stored and transmitted by digital electronic systems. Given its intrinsic nature, binary data often derived from a set of logic operations performed in the logic core of an electronic system and repeated over different

combinations of logic operands. Consequently, a sensible gain in data compression efficiency is achievable if the kernel set of logic operations (logic function) generating the data is known and transmitted/stored in place of the binary data itself. Kolmogorov complexity [16] is a theoretical generalization of this idea: it considers the *shortest* program for a universal computer that outputs the sequence. Unfortunately, Kolmogorov complexity is incomputable [16] limiting its applicability to theory. Nevertheless, relaxing the *shortest* property requirement, the search for such a sub-optimal program (logic function) is still of interest for efficient data compression.

Motivated by this intuition, we study in this section a methodology able to identify a valid logic function that can generate back the initial binary data. Then, such logic function is minimized by logic synthesis techniques effectively eliminating redundancy. We integrate this approach in a lossless compression flow and then we show its reversibility for exact data decompression.

A. Description of a Logic Function Generating a Specific Binary String

The input of general data compression tools is a string of bits, say B. Data decorrelation techniques begin by partitioning B in M sub-blocks $\{S_0, S_1, ..., S_{M-1}\}$ of length $L = \lceil |B|/M \rceil$[1] [6]. We follow this approach to describe a logic function G that outputs B. In this context, the requirement for G is to produce an output S_i when stimulated by the *Binary Representation* (BR) of the partition index i (BR(i) has $N = \lceil log_2(M) \rceil$ bits). More formally, G is an L-output, N-input Boolean function implementing the relation $G(\text{BR}(i)) = S_i$. The original string B can be generated back by stimulating G with consecutive values of BR(i) and concatenating the corresponding output. Note that, the choice for the partitioning number M is dictated by practical considerations on the binary data to be compressed. The same argument also holds for general decorrelation transforms [6].

Algorithm 1 creates a *Sum Of Products* (SOP) representation for G given the partition $\{S_0, S_1, ..., S_{M-1}\}$. Note that G is initially represented in SOP form but can be minimized later in various ways as long as the final logic circuit for G is small. Two promising candidate techniques to minimize G are multi-level synthesis and binary decision diagrams. On the one hand, multi-level synthesis is efficient to manipulate a large amount of primes that are present in the SOP of G for large data. On the other hand, binary decision diagrams based methods can map G in a program with nested if then else to regenerate data. Regardless of what is the best approach, the use of logic synthesis is orthogonal to our compression method enabling a high degree of flexibility to choose the most appropriate minimization technique. The process in Algorithm 1 to generate the SOP for G consists of two nested for loops, the first considers all the L outputs of G while the second one considers all the M partitioned sub-blocks. The rationale is the following: when a sub-block S_i assumes the logic 1 value at the k-th bit, the k-th output of G is updated accordingly by adding the cube BR(i) to its SOP. At the end of this procedure, a valid description for the Boolean function G is obtained.

[1]For the sake of simplicity, fixed-length partitions are considered.

Algorithm 1 G function description.

INPUT: binary strings $\{S_0, S_1, ..., S_{M-1}\}$ (L-bits per each)
OUTPUT: SOP representation for G function
FUNCTION: Construct G($\{S_0, S_1, ..., S_{M-1}\}$)

```
for all k = 0 : L − 1 do
  for all i = 0 : M − 1 do
    if (S_i(k) == 1) then
      add cube BR(i) to SOP for the k-th output of G
    end if
  end for
end for
```

G function example: B = 000001010011000001110111, partition coefficients $M = 8$ ($N = 3$), $L = 3$, partition set $\{S_0, S_1, .., S_7\} = \{\mathbf{0}00, \mathbf{0}01, \mathbf{0}10, \mathbf{0}11, \mathbf{0}00, \mathbf{0}01, \mathbf{1}10, \mathbf{1}11\}$. Consider the first bit highlighted in bold (G_0). The SOP of G_0 is G_0 =BR(6)+BR(7). With BR(i) = $\{I_0, I_1, I_2\}$, the first (0) bit function becomes $G_0 = I_0 I_1 \overline{I_2} + I_0 I_1 I_2$.

By using Algorithm 1, it is possible to describe a function G for every partitioned binary string B. It can be easily verified that the worst case size for the SOP of G is $O(L \cdot M) = O(|B|)$ cubes of N bits each. Thus, the SOP description complexity is linear in the size of the initial string B. However, we want a logic circuit representation for G much smaller than $O(|B|)$ in order to have advantageous compression for B. Logic synthesis techniques are capable to shrink down the size of G preserving its functionality. In the previous example, $G_0 = I_0 I_1 \overline{I_2} + I_0 I_1 I_2$ is minimized in $G_0 = I_0 I_1$ by logic synthesis. On the one hand, optimal synthesis techniques give the best result in terms of synthesized circuit size but require a long runtime. On the other hand, synthesis heuristics produce near-optimal results with efficient runtime. Since data compression targets the size reduction of large files, heuristic techniques are key to have affordable runtime. Then, the capability of synthesis heuristic to produce satisfactory results heavily depends on the initial logic representation. Moreover, there exist logic functions too complex to be recognized and minimized by traditional synthesis heuristics, e.g., arithmetic operations such as exponential, logarithm etc. When one of these complex functions describes the underlying correlation in the data set that we want to compress, the direct synthesis of G via heuristics may reveal to be unfruitful.

We propose here to improve the efficiency the (heuristic) synthesis process by providing additional information about the function G. While G can be fully described by the binary indexes BR(i) for the partitions of B, G also admits some more flexibility exploiting the sequential nature of the decoding process, i.e., G can be stimulated by BR(i) *if and only if* it has been previously stimulated by BR($i-1$). Therefore, introducing $S_{i-1} = G(\text{BR}(i-1))$ as additional input, on top of BR(i), some further simplification for G are enabled. Please note that by introducing S_{i-1} we are not moving to a sequential synthesis approach, instead the synthesis process is made unaware of the provenience of the input S_{i-1} hence remaining in a combinational context. To exploit the S_{i-1} information it is necessary to consider two different cases: (i) $S_{i-1} = G(\text{BR}(i-1))$ is unique in $\{S_0, S_1, ..., S_{M-1}\}$ and (ii)

$S_{i-1} = G(\text{BR}(i-1))$ repeats in $\{S_0, S_1, ..., S_{M-1}\}$. In the first case, the L bits for $S_{i-1} = G(\text{BR}(i-1))$ are sufficient information to uniquely determine $G(\text{BR}(i-1)) = S_i$, consequently the information for S_{i-1} is ORed to the original description of G. In the second case, the information of S_{i-1} has to be ANDed with BR(i) to uniquely determine the output, this condition is not added to G since it does not contain additional (non redundant) information. The updated procedure is presented in Algorithm 2.

Algorithm 2 Synthesis-facilitated description of G.

INPUT: binary strings $\{S_0, S_1, ..., S_{M-1}\}$ (L-bits per each)
OUTPUT: SOP representation for G function
FUNCTION: Construct G($\{S_0, S_1, ..., S_{M-1}\}$)

```
for all k = 0 : L − 1 do
  for all i = 0 : M − 1 do
    if (S_i(k) == 1) then
      add cube BR(i) to SOP for the k-th output of G
      if (S_{i−1} is unique in {S_0, S_1, ..., S_{M−1}}) then
        add cube S_{i−1} to SOP for the k-th output of G
      end if
    end if
  end for
end for
```

The key improvement with respect to Algorithm 1 is the following: if the i-th output for S_i has a unique predecessor S_{i-1} this can be used as alternative (logical or with BR(i)) information to describe G. In other words, we can uniquely determine $G_i = G(\text{BR}(i)) = S_i$ by giving the index i and its binary representation BR(i) but also by specifying the predecessor of S_i, S_{i-1}, provided that it does not repeat in the initial partition. Heuristic synthesis techniques have new minimization opportunities for G, thanks to the additional disjunctive information on $G(\text{BR}(i))$ deriving from S_{i-1} (exploiting the causal nature of G). It is clear that Algorithm 2 can be extended to deal with S_{i-2}, S_{i-3} etc., the choice of the numbers of previous outputs to be employed is then a tradeoff between the effectiveness of the additional information and the increase in representation size. Practical discussion on this topic is given in Section IV.

B. Compression Flow

Our proposed compression flow employs logic synthesis to identify and remove redundancy in binary data. If logic synthesis fails at discovering the underlying function for a certain portion of data, entropy encoding is used to complete the compression process. Procedure details are given in Algorithm 3. First, the input binary string B is partitioned in substrings $\{S_0, S_1, ..., S_{M-1}\}$ as in usual compression flows [6]. A logic function G is then constructed for $\{S_0, S_1, ..., S_{M-1}\}$ using the methodology presented in Algorithm 2. G is synthesized via traditional heuristics considering each output separately. It is worth to stress that even if G has an initial description in SOP (2-level) form, synthesis heuristics can produce general logic networks potentially more compact than 2-level circuits. When the synthesis for a certain output G_i is not effective (it generates time-out or results in a large circuit) then the index of the corresponding input is stored in R for successive treatment. Otherwise, when the synthesis is effective, the logic circuit for G_i is stored in a common optimized logic circuit K. After all the outputs of G have been considered, the logic circuits stored in K may contain redundancy and therefore a sharing extraction algorithm is applied to K to reduce its size. Considering then the indexes in R, they represent difficult functions for which synthesis heuristics have not been able to produce satisfactory (small) logic circuits. We assume that the words (collection of bits) corresponding to such indexes in R have an *uncorrelated nature* and hence are suited to be compressed with entropy encoding techniques rather than to be synthesized in a circuit. Entropy encoding is applied to W (concatenated bits corresponding to the indexes in R). Finally, the compressed string is obtained by concatenating the binary representation for the logic circuit K and the entropy encoded sequence W. The integer set R and the integer numbers $\{M, L\}$ must be provided to the decompression stage in order to fully reconstruct the original string B.

Algorithm 3 Lossless compression of binary string B.

INPUT: Binary string B
OUTPUT: Compressed string C
FUNCTION: Compress B

```
R = ∅ (set of integers – indexes)
K = ∅ (set of logic functions)
W = ∅ (set of bits – string)
partition B in L-bit long {S_0, S_1, ..., S_{M−1}} substrings
construct logic function G for {S_0, S_1, ..., S_{M−1}} (Alg. 2)
for all output i of G do
  synthesize G_i
  if ((synthesis time-out) or (size G_i > threshold)) then
    R ← i
  else
    store G_i's synthesized logic circuit in K
  end if
end for
logic sharing extraction in K
for all i ∈ R do
  for all j = 0 : M − 1 do
    W ← S_j(i)
  end for
end for
C = binary representation of K + entropy encoding of W
```

C. Decompression Flow

The compression method in Algorithm 2 is lossless and therefore exactly reversible. The corresponding decompression method is depicted by Algorithm 4. The input of the decompression method are: the previously compressed string C (logic circuit K + binary string W), the set of integer–indexes R, and the integer values $\{M, L\}$.

The first decompression step is achieved by simulating the logic circuit K with consecutive values of BR(i) in input. Note that the logic circuit K is designed to be stimulated only by incremental values of i, otherwise the functionality of G is lost. If G is described using Algorithm 1 the decoding model

Algorithm 4 Lossless decompression of binary string C.

INPUT: Compressed string C (logic circuit K + binary string W), integer set R, integer values $\{M, L\}$
OUTPUT: Original string B
FUNCTION: Decompress C

$X = \emptyset$ (set of bits – string)
$Y = \emptyset$ (set of bits – string)
for all $i = 0 : M - 1$ **do**
  $X \leftarrow K(\text{BR}(i))$
end for
$Y \leftarrow$entropy-decode(W)
$B =$ interleave X and Y according to $\{R, L\}$

is just a combinational logic circuit producing as output the (partial) strings $\{S_0, S_1, ..., S_{M-1}\}$ in sequence. Otherwise, when Algorithm 2 is employed to facilitate the synthesis of G, the decoding model needs to be updated. The Mealy FSM model in Fig.1 is a valid extension to decompress logic circuits described by Algorithm 2. The previous output S_{i-1} becomes

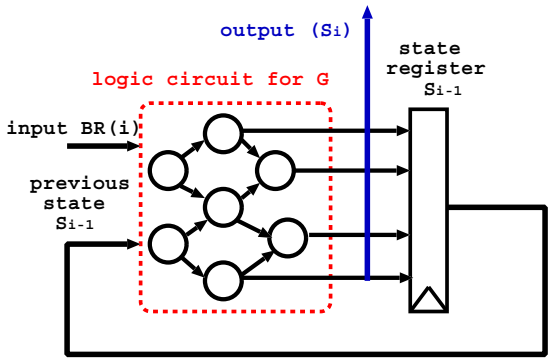

Fig. 1. Decompression FSM (Mealy) model.

the state of the Mealy machine and together with the current input BR(i) determines the output value. Denote by X the concatenation of binary outputs produced by the consecutive simulation of G (either FSM or combinational model).

The second decompression step is perfomed via entropy decoding of the binary string W, produced by Algorithm 3. Denote by Y the entropy decoded string.

The final step to recover the original string B consists of interleaving strings X and Y. Indeed, the initial partition $\{S_0, S_1, ..., S_{M-1}\}$ can be built back from X and Y by (i) partitioning in M sub-strings X and Y and (ii) merging pair-wise each X_i, Y_i sub-strings in unique L-bit long sub-string $\{X_i, Y_i\} \rightarrow S_i$. Such merged substring S_i has Y_i elements placed in the indexes pointed by the R set and X_i elements placed consecutively in the rest of the positions. At the end of this procedure the initial partition $\{S_0, S_1, ..., S_{M-1}\}$ is obtained, B follows by direct concatenation of S_i strings one after the other.

X-Y interleaving example: $X = 000111010$, $Y = 101$, $M = 3$ ($N = 2$), $L = 4$, $R = \{2\}$. Given M (number of partitions) X and Y can be seen as $X = \{000, 111, 010\}$, $Y = \{1, 0, 1\}$. R says that the 2^{nd} index of each partition originally belongs to Y while, by duality, the rest of indexes belong to X. Consequently, $B = \{0100, 1011, 0110\}$ which corresponds to $B = 010010110110$.

Note that the overall decompression technique requires M simulations of the logic circuit for G, plus M interleaving operations. Thus, it can be shown that the decompression runtime is $O(M \cdot |G|)$, where $|G|$ represents the size of the logic circuit G to be simulated.

IV. Experimental Results

In this section, we evaluate the advantage of the proposed lossless compression technique. Decompression runtime is not reported for brevity. Comparisons with state-of-art lossless compression tools are also given.

A. Methodology

The top module of the lossless compression method proposed is described by Algorithm 3. Then, its core constituents are (i) algorithm(s) for G description, (ii) logic synthesis heuristics to minimize G and (iii) entropy encoding for the string portion not represented by G. The top module is implemented in PERL language and manages the interaction between the three major sub-modules. Algorithm 2 for G description is implemented in C language. Only the previous value S_{i-1} is considered in the implementation of Algorithm 2 since, for the considered data sets, the use of additional previous values do not carry synthesis improvements but representation size overhead. Logic synthesis is accomplished using ABC [17] academic synthesis tool. Entropy encoding for the portion of the input string not represented by G is achieved via the standard-de-facto ZIP tool [18].

In order to validate the proposed compression method, we consider data set benchmarks deriving from causal processes: 1) a perfect line measurement, 2) a line measurement affected by white noise, 3) a parabolic measurement and 4) apparently random data but automatically generated by a XOR-intensive logic circuit. Benchmarks 1-3 are data-set observable in physics experiments (Electronics, Mechanics, Astronomy etc.) while benchmark 4 represents the output of a computer program, or integrated-circuit, running over several operation cycles. In all 4 cases, we consider only data sets larger than 1 MB. The focus on such particular data sets, in place of the general benchmark suite, is motivated by the nature of the proposed compression, which is intended, and designed, for high-correlated data sets.

For these benchmarks (still binary strings B), the number of partitions considered (M) is $\lceil |B|/20 \rceil$. Therefore, G in our experiments is a 20-output, $\lceil log_2(\lceil |B|/20 \rceil) \rceil$-input Boolean function. The synthesis timeout threshold in Algorithm 3 is set as 1.0s while the size threshold is 10% of the initial logic representation size.

The counterpart compression tools considered are: (i) ZIP [18] (based on LZ [8] technique), (ii) DCT transform + ZIP, (iii) bzip2 [19] (based on Burrows-Wheeler transform [9]) and (iv) 7zip [20] (evolutions of LZ [8] technique).

B. Results

Table I shows data compression results. Among all the compression techniques considered, our approach shows the

TABLE I
DATA COMPRESSION RESULTS

Benchmark	Original Data Size	ZIP	DCT+ZIP	bzip2	7zip	Our Approach	ZIP Runtime
Linear Data	2.2 MB	208 KB	868 KB	316 KB	60 KB	8 KB	0.3 s
	25 MB	2.1 MB	8.3 MB	3.1 MB	888 KB	8 KB	2.1 s
	287 MB	21 MB	81 MB	31 MB	3.4 MB	302 KB	32 s
Linear Data + Noise	2.2 MB	264 KB	872 KB	258 KB	212 KB	80 KB	0.4 s
	25 MB	2.7 MB	8.4 MB	2.6 MB	2.4 MB	700 KB	3.0 s
	287 MB	27 MB	84 MB	30 MB	23 MB	7.1 MB	43 s
Quadratic Data	3.3 MB	484 KB	816 KB	532 KB	272 KB	8 KB	1.0 s
	39 MB	5.3 MB	7.6 MB	6.1 MB	3.3 MB	16 KB	6.1 s
	449 MB	59 MB	71 MB	67 MB	40 MB	566 KB	64 s
Random (XOR-intensive network)	1.6 MB	116 KB	304 KB	124 KB	44 KB	8 KB	0.1 s
	20 MB	1.2 MB	3.2 MB	1.5 MB	796 KB	8 KB	1.2 s
	230 MB	12 MB	31 MB	15 MB	3.8 MB	234 KB	10 s
Average runtime (normalized to ZIP)	–	1	–	1.5 x	8 x	12 x	–

highest compression gain ranging from 1 to 3 order of magnitues. The best compression ratio among counterpart tools is achieved by 7zip, being still 1$\sim$2 order of magnitudes smaller than our approach. The considerable compression gain improvement of our method comes from the capability of logic synthesis to recognize (by eliminating redundancy) the core function underlying in the data we want to compress. For example, the first benchmark representing linear data is reduced to the function $G(\text{BR}(i))$=BR(i), which corresponds to a logic circuit where input and output are just connected by a wire. Then, the third benchmark (square function) is reduced to the function $G(\text{BR}(i))$=BR(i) + $G(\text{BR}(i-1))$ (where + is the binary addition operation), which corresponds to a two-operands binary adder circuit. It is then obvious why such large compression improvement is possible: we store only the basic function (or program) generating the data while other traditional compression tools cannot rely on this opportunity. About the runtime, ZIP is the fastest tool while our approach is on average 12x slower. This is due to the current implementation of the compression software calling external synthesis (ABC) and entropy encoding tools (ZIP). We expect that the runtime can be improved in a fully integrated software.

It is interesting to notice that in the data set 2 (linear data + noise) our proposed approach is able to identify the random portion of the data and then isolate it. Indeed, for the noisy bits of the partitions, the synthesis process always produced timeout. Then, the identified complex bits are treated as uncorrelated and compressed with entropy-encoding technique. As a result, the size of the compressed data tends to the size of the random noise superposed on the linear data.

The largest data set compressed is 449 MB evidencing the scalability of our method. This is thanks to the $O(|B|)$ size of the initial G description and to the efficiency of logic synthesis heuristics employed (and-inverter-graphs based techniques in ABC [17]).

V. CONCLUSIONS

Resource usage is a key factor of today's software and hardware applications. To reduce resource usage, in terms of data storage or transmission capacity, lossless data compression techniques are widely employed. In this paper, we use logic synthesis to compact the size of causal data sets enabling novel lossless compression opportunities. In the data compression context, the capability of modern logic synthesis tools to identify and remove redundancy, while preserving the initial behavior of the circuit, paves the way for a general lossless compression approach with the aim to find the underlying logic function generating the input data. Experimental results on data sets derived from causal processes show that our compression method based on logic synthesis reduces by 1$\sim$2 orders of magnitudes the data size compared to state-of-art compression tools such as ZIP, bzip2 and 7zip.

REFERENCES

[1] S. Nassif, *From Circuits to Cancer*, Keynote, ASPDAC 2013.
[2] F. Liu, B.R. Hodges, *Dynamic River Network Simulation at Large Scale*, Proc. DAC 2012.
[3] P. Lin, S.P. Khatri, *Application of Logic Synthesis to the Understanding and Cure of Genetic Diseases*, Proc. DAC 2012.
[4] Byron Washom, *Smart Grid for the 21st Century*, ICCAD 2009.
[5] M. Hilbert, P. Lopez, *The World's Technological Capacity to Store, Communicate, and Compute Information*, Science 332 (6025): 60-65, March 2013.
[6] K. Sayood, *Introduction to Data Compression*, 3rd ed., Elsevier, 2006.
[7] C. Lefurgy *et al.*, *Improving code density using compression techniques*, 30th ACM/IEEE Proc. on Microarchitectures, 1997.
[8] J. Ziv, A. Lempel, *A Universal Algorithm for Sequential Data Compression*, IEEE Trans. on Information Theory 23 (3): 337-343, May 1977.
[9] M. Burrows, D. Wheeler, *A block sorting lossless data compression algorithm* Technical Report 124, Digital Equipment Corporation, 1994.
[10] Joint Photographic Experts Group standard, http://www.jpeg.org.
[11] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.
[12] D.A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, PhD thesis, MIT, 1952.
[13] J. Augustinea, *et al.*, *Switching theoretic approach to image compression*, Signal Processing, 44, 243-246, 1995.
[14] D. Pramanik, *et al.*, *Lossless compression of images using minterm coding*, Proc. ICICS, 1997.
[15] J. Yang, *et al.*, *Lossless compression using 2-level and multilevel Boolean minimization*, Proc. SIPS, 2006.
[16] A.N. Kolmogorov, *On Tables of Random Numbers*, Theoretical Computer Science 207 (2): 387395, 1963.
[17] ABC Synthesis Tool [Online] http://www.eecs.berkeley. edu/alanmi/abc/
[18] ZIP Compression Tool [Online] http://www.pkware.com/
[19] bzip2 Compression Tool [Online] http://www.bzip.org/
[20] 7ZIP Compression Tool [Online] http://www.7-zip.org/