Squashing Microcode Stores to Size in Embedded Systems while Delivering Rapid Microcode Accesses

Chengmo Yang, Mingjing Chen, and Alex Orailoglu

Computer Science and Engineering Department
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
{c5yang, mjchen, alex}@cs.ucsd.edu

ABSTRACT

Microcoded customized IPs offer superior performance and direct programmability of micro-architectural structures compared to instruction-based processors, yet at the cost of drastically enlarged code sizes. Code compression can deliver size reductions but necessitates attention to performance issues, so that the performance benefits of microcoded IPs are not squandered in the process. To attain this goal, we propose in this paper a fast code compression technique through exploiting the fact that the microcodes contain a sizable amount of unspecified bits. Although the values and the positions of the specified bits are highly irregular, the proposed technique can still flexibly and precisely fill in these fully specified bits through utilizing a linear network. The linear property inherent in the compression strategy in turn enables the development of an extremely low-overhead decompression engine. At runtime, the decompressed code can be generated in such a way that all the specified bits can be filled as required by a fixed-bandwidth XOR network. The combination of the proposed flexible XOR-based network with a minimum two-level storage for highly specified fields, such as immediate values, offers utmost code compression, attained within a negligible amount of performance and hardware overhead.

Categories and Subject Descriptors:

B.1.5 [Hardware]: –Microcode Applications

General Terms:

Performance, Design

Keywords:

Microcode compression, Microcoded processors

1. INTRODUCTION

Shrinking time-to-market and high demand for productivity has driven the use of microcoded customized IPs in embedded system design. Examples of such design platforms, typically referred to as Horizontal Microcoded Architectures, include PICO [1], ARM OptimoDE [2], TIPI [3], NISC [4], and FlexCore [5]. Compared to instruction-based

processors, microcoded customized IPs directly expose the entire control of microarchitectural components to the compiler, thus allowing the compiler to directly exploit parallelism across functional units without instruction abstraction. As applications can be directly compiled to horizontal microcodes, such architectures can be designed without costly decoders, controllers and hardware schedulers, thus offering superior performance, lower power, and lower area than conventional RISC processors.

Despite the aforementioned benefits, the code size of microcoded IPs is drastically enlarged. Storing microcodes directly on-chip requires large memory blocks that impose significant area and power overhead. As on-chip memory is one of the most limiting resources in cost-sensitive embedded systems, microcoded IPs necessitate aggressive code compression techniques. Meanwhile, to ensure that the performance benefits of microcoded IPs are not squandered as a result of code compression, the on-line decompression engine should provide high throughput instruction flow within highly constrained latency. Compression techniques based on variable-length encoding [6, 7] thus are not desirable for such systems, as these approaches require more complex decoders, incurring costs in performance, area and power in addition to long design and verification times. Dictionary-based compression techniques are also not desirable, as these techniques incur significant overhead either in storing the large dictionary or in constantly reloading the dictionary.

An important property of horizontal microcodes is that each microcode typically contains a sizable number of unspecified bits (denoted as ‘X’). These X bits correspond to the control signals of the functional units that are idle at a given cycle, thus enabling these bits to be mapped to either 0’s or 1’s in the final executable binary without affecting program behavior. This flexibility can be exploited to attain a dictionary-free fixed-length encoding technique. Specifically, in this paper we propose an XOR network-based compression technique that sets the ‘X’ bits in such a way that the values form a linear relationship with the specified bits. The linear relationship thus ensures that the fully specified bits can be precisely filled, although the values and the positions of these bits are highly irregular. Meanwhile, the linear relationship among the various bit positions also enables the development of an extremely low-overhead decompression engine, composed of only a fixed-bandwidth XOR network.

The effectiveness of the proposed compression technique is determined by the quality of the linear network as well as the minimum number of unspecified bits in a microcode. To improve the compression ratio, we furthermore propose

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS’09, October 11–16, 2009, Grenoble, France.

Copyright 2009 ACM 978-1-60558-628-1/09/10 ...\$10.00.

a set of optimization techniques to maximally minimize the correlation of the linear network and to balance the number of ‘X’ bits across various codewords. The combination of the proposed flexible XOR network with a minimum on-chip storage for highly specified fields, such as immediate values, offers utmost code compression, attained within a negligible level of performance and hardware overhead.

The rest of the paper is organized as follows. Section 2 briefly reviews code-size reduction techniques for embedded processors and microcoded IPs. Section 3 discusses in detail the proposed linear network-based compression technique, while the techniques for improving the compression ratio are discussed in Section 4. The efficacy of the proposed compression technique is experimentally verified in section 5, and Section 6 offers a brief set of conclusions.

2. TECHNICAL BACKGROUND

As embedded systems typically impose a strict cost and power budget, a number of code-size-reduction techniques have been proposed for embedded processors. These techniques can be briefly categorized in three groups, including *compiler-based optimizations*, such as interprocedural optimization and procedural abstraction of repeated code fragments [8, 9], *ISA modifications*, such as the Thumb [10] and the MIPS16 [11] instruction sets, and *code compression techniques*.

In code compression techniques, the executable program is compressed offline and decompressed on-the-fly during execution [6]. These techniques are more desirable for embedded cores and microcoded IPs as compared to the first two categories. This is because the compiler optimization and ISA modification approaches necessitate modifications on compilers and/or linkers, while in code compression, the compression and decompression are typically performed in a manner transparent to the processor and the compiler. The efficiency of a code compression scheme can be measured by the *compression ratio*, defined as follows:

$$\text{Compression Ratio} = \frac{\text{Compressed code size}}{\text{Original code size}} \quad (1)$$

Compared to general purpose data compression approaches, code compression exhibits extra challenges for handling the control altering instructions, such as branch, jump, and return instructions. It is thus essential to ensure that each instruction can be decompressed *individually* with no reliance on any preceding instructions. Moreover, it is also necessary to establish a mapping between the original address space and the compressed address space so that upon a control altering instruction, the address of the target instruction can be easily recalculated. This requirement is more challenging for *variable-length* encoding techniques, wherein the mapping between the two address spaces is irregular.

Compression techniques based on variable-length encoding exploit the uneven occurrence ratio of instructions to attain code size reduction. For example, in [6], the Compressed Code RISC Processor (CCRP) based on Huffman encoding, is proposed. Unique instructions in the program are stored in a dictionary, wherein the indices of the instructions are determined by Huffman coding. In IBM Codepack [7], each 32-bit instruction is evenly split into two parts, while two dictionaries are used to record the unique patterns of each part. This two-dictionary architecture is fur-

thermore extended in [12], wherein the instruction bits are rearranged so as to balance the dictionary sizes.

While the aforementioned techniques can effectively improve the compression ratio, the decompression speed is typically quite slow due to the variable sizes of compressed instructions. To minimize the decompression overhead, these variable-length encoding techniques usually rely on the existence of a cache, so that decompression can be performed only upon a cache miss. These architectures are denoted as *pre-cache* decompression. Compression thus does not improve the utilization of the cache. Moreover, as missed instructions do not reside at the same address in the cache as in memory, a line address table (LAT) table [6] is needed to record the mapping between the compressed and the original address spaces. This extra overhead, together with the necessity of caches, makes these variable-length compression techniques less desirable for cost-sensitive embedded systems, especially microcoded IPs.

To attain fast decompression in a no-cache embedded system, dictionary-based fixed-length encoding approaches are usually employed. The basic idea is to store all the unique microinstructions into a lookup table (LUT). The compression ratio thus is determined by the number of entries in the LUT, which is in turn determined by the number of unique codewords within a program. Obviously, the attainment of low compression ratio relies on the existence of high redundancy and repetition of codewords in the program. As two distinct codewords may partially share a sequence of values in common, researchers have also proposed to vertically partition each codeword of a program into multiple groups [13, 14] to maximally exploit the potential repetition. Although the compression ratio is improved, the required LUT size is still quite large. It has been reported in [15] that for the EEMBC benchmarks, the technique proposed in [13] requires a LUT with 300 – 400 entries. Such a large LUT thus imposes significant hardware overhead and access latency onto the target embedded processor or microcoded IPs. To reduce the LUT size, researchers have also proposed to dynamically reload the content of the LUT using dedicated table manipulating instructions [15]. However, such instructions will not only increase the static code size but also incur performance overhead during program execution.

3. LUT-FREE FIXED-LENGTH MICROCODE COMPRESSION

The use of a linear network to compress microcodes is motivated by the observation that each microcode typically exhibits a sizable number of unspecified bits. As a single microcode is composed of the signal bits for controlling the functional units, these X bits correspond to the control signals of the functional units that are idle at a given cycle. Examples include register-file read and write addresses, MUX selection, and ALU operation signals.

Although the unspecified bits can be flexibly filled as either 0’s or 1’s in the final executable binary, this flexibility has not been fully exploited by the traditional LUT-based compression approaches. As these approaches exploit the compactability between codewords to reduce code size, the compression ratio is constrained by the number of bit conflicts. As an example, the following two codewords need to be captured in distinct entries in the LUT, even if each codeword contains only a single specified bit.

$C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 \rightarrow X_0 X_1 X_2 X_3$
0 1 x x x x 1 0 $\rightarrow$ 0 0 1 0
1 1 0 1 x x x x $\rightarrow$ 0 1 1 0
x x 0 1 1 0 x x $\rightarrow$ 1 0 1 0
x x x x 1 1 1 1 $\rightarrow$ 0 0 0 1
x x 0 0 x x 1 1 $\rightarrow$ 0 1 1 1
1 0 0 0 x x x x $\rightarrow$ 1 0 1 1
1 1 x x 0 1 x x $\rightarrow$ 1 0 0 1
x x x x 1 1 0 0 $\rightarrow$ 0 1 1 0

Figure 1: Compression Example: 8-bit microcodes to 4-bit seeds

Figure 2: Linear network-based hardware decompressor

$$\begin{aligned}
C_0 &= X_0 \oplus X_1 \\
C_1 &= X_0 \oplus X_2 \\
C_2 &= X_0 \oplus X_1 \oplus X_2 \\
C_3 &= X_2 \oplus X_3 \\
C_4 &= X_0 \oplus X_1 \oplus X_3 \\
C_5 &= X_1 \oplus X_3 \\
C_6 &= X_1 \oplus X_2 \oplus X_3 \\
C_7 &= X_0 \oplus X_3
\end{aligned}$$

Figure 3: Decompression equations

$$\begin{array}{cccccccc}
x & x & x & x & x & x & 1 & x & x & x & x & x \\
x & x & x & x & x & x & 0 & x & x & x & x & x
\end{array}$$

Instead of exploiting the compactability between codewords, the proposed linear network-based compression technique directly exploits the flexibility in filling the unspecified bits in each microcode. Using a set of linear transformations, not only the values but furthermore the bit positions of these specified bits can be captured in a compact form. In this way, these specified bits can be flexibly and precisely filled, even though the distribution of them throughout the codeword may display high levels of irregularity.

An illustrative example of the proposed compression technique is shown in Figure 1, wherein a set of 8-bit codewords, each of which contains 4 specified bits, are compressed into 4-bit seeds. The corresponding XOR network and the linear equations are shown in Figures 2 and 3, respectively.

3.1 XOR network-based Code Compression

An N -to- M XOR network receives as inputs an N -bit seed, and generates an M -bit fully specified code through performing M groups of XOR operations on the seed bits. Obviously, such a network delivers a compression ratio of N/M .

Since all the XOR operations performed in the network are linear operations, the transformation from the N -bit seed to the M -bit codeword also constitute a linear transform, with each of the M -bit codewords being a linear combination of an N -bit seed. Moreover, such a linear transformation can be characterized by the structure of the network, which can be furthermore represented by an $M \times N$ coefficient matrix, wherein a '1' in the matrix represents the existence of a connection between the output and input bits of the XOR network. As an example, the matrix representation of the compression structure in Figure 2 is shown in Equation (2).

$$\begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3
\end{bmatrix}
=
\begin{bmatrix}
C_0 \\
C_1 \\
C_2 \\
C_3 \\
C_4 \\
C_5 \\
C_6 \\
C_7
\end{bmatrix}
\quad (2)$$

To generate the specified bits in the original code, the set of linear equations, which describe the linear combination relationship among these bits, needs to be solved. Depending on the positions of the specified bits in the codeword, a

subset of the rows of Equation (2) is utilized for the solution. For example, in the first codeword in Figure 1, only C_0 , C_1 , C_6 and C_7 are specified, with the corresponding values being $[0 \ 1 \ 1 \ 0]$. The linear equation system for this code thus can be specified as follows:

$$\begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3
\end{bmatrix}
=
\begin{bmatrix}
C_0 \\
C_1 \\
C_6 \\
C_7
\end{bmatrix}
=
\begin{bmatrix}
0 \\
1 \\
1 \\
0
\end{bmatrix}
\quad (3)$$

These equations can be solved through the utilization of any Gauss-Jordan elimination [16] methodology. If such a linear equation system has at least one solution, the corresponding codeword thus can be successfully compressed.

Due to the irregular distribution of the specified bits, distinct codewords necessitate solution of different sets of linear equations. Yet the likelihood of finding a seed for each linear system is quite high as long as the number of specified bits in each code (the number of equations in the system) remains small. In fact, compression is guaranteed to be successful if the number of specified bits in a codeword does not exceed the *rank* of the coefficient matrix, since in this case there exists at least one solution in such a linear equation system. On the other hand, if the number of specified bits exceeds the rank of the matrix, the existence of a solution is determined by the values of the specified bits.

To maximally preclude the generation of an unsolvable linear system, the *root cause* for linear equation insolvability needs to be examined. If we incorporate the column vector of the specified bits into the coefficient matrix, an *augmented* matrix with a size of $M \times (N + 1)$ can be constructed. A comparison between the *augmented* matrix and the *coefficient* matrix indicates that no solution exists for this linear system if and only if the rank of the augmented matrix exceeds the one of the coefficient matrix. An illustrative example is shown in Figure 4, wherein the three rows in the coefficient matrix are *linearly dependent* such that any row can be generated through XORing the other two rows. In contrast, in the augmented matrix this linear dependency disappears, thus forcing the augmented matrix to display a larger rank than the coefficient matrix.

The occurrence frequency of the unsolvable cases strongly depends on the linear dependencies among the XOR equations, that is, the rows in the coefficient matrix. A high correlation among these rows would greatly reduce the rank of the coefficient matrix, thus increasing the likelihood of the insolvability condition outlined above. Therefore, the

$$\begin{array}{c}
\text{Coefficient} \\
\text{matrix} \\
\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
\text{rank} = 2
\end{array}
\quad
\begin{array}{c}
\text{Augmented} \\
\text{matrix} \\
\begin{bmatrix} 1 & 1 & 0 & \color{red}{1} \\ 1 & 1 & 1 & \color{red}{0} \\ 1 & 0 & 1 & \color{red}{0} \end{bmatrix} \\
\text{rank} = 3
\end{array}$$

Figure 4: No solution case

Figure 5: Partition group generation: 3 PGs, each contains 5 disjoint partitions of 3 bits

construction of a low correlation XOR network is essential for the effectiveness of the proposed compression scheme.

3.2 XOR network correlation minimization

An important property of the proposed XOR network is the existence of common input bits among distinct output functions, as the network, used as a decompressor, should produce more outputs than the number of inputs. However, a large overlap of input bits may increase the correlation intensity of the network, which may in turn degrade the attainable compression ratio. In order to maximally reduce linear dependencies and deliver an appreciable compression ratio, a *1-degree overlap* constraint can be imposed during the process of network construction. Here, the parameter *overlap degree* is defined as the maximum number of common bits between any two XOR functions.

Although the 1-degree overlap constraint drastically diminishes linear dependencies, it on the other hand sharply reduces the number of attainable XOR functions (output bits). We therefore need to develop an algorithm to maximally identify a set of input combinations that satisfy the strict overlap constraint.

If each output bit in the linear network is generated by an S -input XOR function, the set of N input bits can be partitioned into $\lceil N/S \rceil$ disjoint partitions. These disjoint partitions constitute a *partition group*, denoted as PG . Obviously, the disjointness directly implies that the *overlap degree* of any two partitions within a single PG is consistently 0. The problem of forcing *overlap degree* to be 1 thus translates to the identification of a maximum number of PG s such that for any two partitions in distinct PG s, the overlap degree is 1.

The aforementioned type of partition groups can be constructed using the deterministic partitioning strategy proposed in [17]. Specifically, we use $P(k, b, i)$ to denote a specific instance of input selection for the i^{th} element in the b^{th} partition of the k^{th} partition group. It has been shown in [17] that the following shuffling strategy can be used to construct partition groups that satisfy the 1-degree overlap constraint.

$$P(k, b, i) = P(0, b \oplus (k \otimes i), i) \quad (4)$$

Equation (4) specifies the manner of constructing the k^{th} partition group through linearly shuffling elements across the multiple partitions in the original partition group. To satisfy the 1-degree overlap constraint between any two partitions in any two PG s, the operation $b \oplus (c \otimes i)$ needs to be a *bijective function*, i.e., one-to-one and onto. For a partition group with a size of B (i.e. it contains B disjoint partitions), the addition and multiplication operations defined in the Galois field $GF(B)$ can be used to construct the bijective shuffling function. If B is a prime number, a straightforward

implementation can be attained through modulo- B addition and multiplication.

To concretely illustrate this shuffle function in satisfying the *1-degree overlap* requirement, Figure 5 presents an example for a 15-input XOR network construction. As each output bit is generated by XORing 3 input bits, the 15-bit inputs are evenly partitioned into 5 groups of 3 bits, implying that $B = 5$ and $S = 3$ for this particular example. As the size of the partition group, 5, is a prime number, the modulo-5 addition and multiplication operations can be utilized for constructing the various partition groups. The two sets of elements highlighted in P_0 , $(0, 4, 8)$ and $(0, 7, 14)$, are selected to form one partition in P_1 and one in P_2 , respectively. As can be seen, the shuffling function selects only one and exactly one element from each partition in P_0 , thus fulfilling the 1-degree overlap requirement. For this example, a maximum number of 5 partition groups can be generated. The entire set of these groups is listed in Table 1, wherein each row corresponds to a partition group PG_i and each box represents the three inputs that are used to generate one output bit. The disjointness within each partition group and the single overlap property between distinct partition groups can be easily verified from the table.

The aforementioned XOR network construction approach can generate a total number of B partition groups, each of which contains B partitions. As a result, the proposed technique can generate a maximum number of B^2 outputs that satisfy the 1-degree overlap constraint. Since each output needs to be driven by a linear equation, the value of B and hence, the value of S can be determined according to the values of N and M , that is, the output and input bandwidth of the XOR network. This relationship is formally specified in the following equation:

$$S = \lceil N/B \rceil \quad (5a)$$

$$B^2 \geq M \quad (5b)$$

The network shown in Table 1 can generate up to 25 output bits using 15 inputs, implying that the lower bound of the compression ratio is 0.6. More formally, the ratio of (5a) over (5b) indicates that the compression ratio of this XOR network, defined as N/M , is constrained by the value S/B .

	P_0	P_1	P_2	P_3	P_4
PG_0	0 1 2	3 4 5	6 7 8	9 10 11	12 13 14
PG_1	0 4 8	3 7 11	6 10 14	9 13 2	12 1 5
PG_2	0 7 14	3 10 2	6 13 5	9 1 8	12 4 11
PG_3	0 10 5	3 13 8	6 1 11	9 4 14	12 7 2
PG_4	0 13 11	3 1 14	6 4 2	9 7 5	12 10 8

Table 1: 15-input XOR network construction ($S = 3$, $B = 5$)

The attainment of a lower compression ratio thus requires a smaller value of S . A smaller value of S also implies that the network needs fewer numbers of XOR gates, which in turn reduces the hardware and performance cost of the network. However, a smaller value of S degrades the randomness of the network, as the network construction approach delivers a bit overlap ratio of $1/S$. A set of experiments indicates that generating each output by XORing 3 or 4 inputs provides sufficient randomness, thus resulting in the value of S being set to 3 or 4 during the network construction process.

4. COMPRESSION RATIO ENHANCEMENT

An important aspect of the proposed XOR network-based microcode compression technique is that the attainable compression ratio is determined by both the positions and the total number of unspecified bits within a codeword.

Given a fixed number of unspecified bits, the attainable compression ratio of a codeword is determined by the positions of these bits. Specifically, a successful compression of all the codewords needs to ensure the existence of at least a single ‘X’ bit within each set of *linearly dependent* bit positions. This goal can be accomplished through manipulating the positions of the ‘X’ bits, thus motivating the proposal of a *column reordering* technique.

As the ‘X’ bits within a program typically exhibit a highly unbalanced distribution across codewords, the various codewords in turn exhibit a nonuniform compression ratio. However, the overall compression ratio of a fixed-length compression scheme is determined by the codeword with the worst compression ratio. It is therefore preferable to increase the ratio of the ‘X’ bits in the *hard-to-compress* codewords so as to enhance the overall compressibility. Given the fact that the number of ‘X’ bits in a codeword cannot be arbitrarily enlarged, an increased ‘X’ bit ratio can only be attained through *reducing* the number of specified bits in the hard-to-compress codewords. We propose **two** techniques to attain this reduction: a *column merging* technique and a *hybrid compression* approach.

In the following subsections, the three compression ratio enhancement techniques outlined above as well as the overall code compression flow will be discussed in detail.

4.1 Column reordering

As discussed in Section 3.1, if a set of linearly dependent bits are all specified in a codeword, the codeword might be incompressible as this linear dependency reduces the rank of the coefficient matrix and possibly leads to a rank mismatch between coefficient matrix and the augmented matrix. To maximally preclude such an unpalatable situation, at least one ‘X’ needs to be inserted in each set of *linearly dependent* bit positions so as to break the linear dependency.

In the proposed XOR-network construction technique, each partition group, composed of B partitions, covers every input of the XOR network exactly once. Therefore, the output functions corresponding to any two partition groups will be linearly dependent. Maximal preclusion of the insolvability condition necessitates the inclusion of at least one ‘X’ bit in any two partition groups. As each partition group maps to B consecutive output bits of the XOR network, at least one ‘X’ bit thus needs to be included in these B consecutive columns in the codeword. To attain this goal, we exploit the flexibility in reordering the columns of microcodes. This flexibility is provided by custom IPs that only hold a highly limited

number of applications. The design can be customized in such a way that the original control sequence is attained through rerouting the outputs of the decompressor.

In the proposed reordering algorithm, in order to simultaneously fulfill the requirement of at least one ‘X’ in every B consecutive columns for all the codewords, the current length of the specified bit sequence of each codeword needs to be recorded during the reordering process. The detailed reordering algorithm is shown in Algorithm 1, wherein the length of the specified bit sequence of codeword j is denoted as $length(j)$. These values are updated upon the selection of a new column k , as shown in lines 11 – 17. To select a suitable column from the remaining set $CoList$, all the codewords of which $length(j) = B - 1$ are examined. A column k is marked as *suitable* if it contains X’s in all these codewords. Finally, among all the *suitable* columns, the one with the minimum number of X’s is selected. This column reordering process is concretely shown in lines 4 – 9.

Algorithm 1 Column reordering algorithm

```

1:  $CoList \leftarrow \{\text{all columns}\};$ 
2:  $max\_length \leftarrow 0;$ 
3: while  $CoList \neq \phi$  do
4:   if  $max\_length < B - 1$  then
5:     Select the first column  $k$  in  $CoList$ ;
6:   else
7:     Identify all the columns  $i \in CoList$  that contains a
       ‘X’ in codeword  $j$ , if  $length(j) = B - 1$ ;
8:     Among this set, select column  $k$  with the minimum
       number of X’s;
9:   end if
10:   $CoList \leftarrow CoList - \{k\};$ 
11:  for codeword  $j = 0$  to  $C_{max}$  do
12:    if  $code(j, k) = \text{‘X’}$  then
13:       $length(j) \leftarrow 0;$ 
14:    else
15:       $length(j) \leftarrow length(j) + 1;$ 
16:    end if
17:  end for
18:   $max\_length \leftarrow max_j(length(j));$ 
19: end while

```

4.2 Column merging

The number of specified bits in the *hard-to-compress* codewords can be reduced through exploiting column merging opportunities, induced by the redundancy in codewords. As customized microcoded IPs are typically developed to hold a set of applications, a particular program may not make full utilization of the provided resources. For example, the IP may provide a divider that may not be used by a particular application. Similarly, the IP may provide 16-bit immediate values, among which only 10 bits are utilized by a particular application. As a result of this resource underutilization, the values in certain columns of the horizontal microcodes, such as the divider control signals and the most significant bits of the immediate field, will be highly similar. Each of these highly similar columns in the microcodes thus do not need to be driven directly by the decompressor. Instead, they can be concurrently driven in a broadcasting manner by a single output bit of the decompressor.

Fundamentally, two columns in the microcodes are considered to be *strictly* compatible if their values in each code-

Figure 6: Reducing specified bits through column compaction

word are identical. As an example, columns C_0 and C_3 in Figure 6 are strictly compatible as they display identical values in each codeword. These columns can be merged in a greedy manner, as the strict compatibility relationship is *transitive*. However, as this strict compatibility requires complete identity, the number of strictly compatible columns is usually quite limited in programs with a large number of codewords. In this situation, further merge opportunities can only be identified through exploiting the compatibility between a specified bit and an ‘X’ bit. Two columns are thus considered to be *loosely* compatible if they do not display specified yet distinct values in any codewords. According to this criterion, it can be easily checked that in Figure 6 there exist 6 pairs of *loosely* compatible columns.

As the column merging approach just described may reduce both the number of specified bits and the number of unspecified bits, it should be selectively applied. Specifically, in order to retain all the ‘X’ bits in the *hard-to-compress* codewords while maximally reducing the specified bits in them, a column should be precluded from being merged with other columns if it contains an ‘X’ bit in a *hard-to-compress* codeword. In Figure 6, columns C_1 and C_2 respectively contain ‘X’ bits in the second and the sixth codewords that are considered to be *hard-to-compress*. These columns, although they are loosely compatible, are thus precluded from being merged. On the other hand, the consumption of ‘X’ bits in the other codewords is allowed, since this consumption does not impact the overall compression ratio that is limited by the *hard-to-compress* codewords. As can be expected, this compaction strategy thus results in a much more balanced ‘X’ bit distribution among the codewords (e.g., the compacted microcode shown in Figure 6), which in turn delivers an enhanced overall compression ratio.

In the proposed work, a codeword is considered as *hard-to-compress* if it contains a smaller number of ‘X’ bits than a predetermined threshold. For an XOR-network with N inputs and M outputs (thus offering a compression ratio of N/M), this threshold can be set to the value $(M - N)$.

4.3 Hybrid compression approach

A comparison between the proposed XOR network-based compression and the standard LUT-based compression indicates that the former can effectively compress the columns with a balanced number of unspecified bits, while the latter can effectively compress the columns with highly clustered values. In light of this observation, we thus propose a hybrid approach which, based on a functional decomposition of the microwords, combines the advantages of both compression schemes to attain an improved overall compression ratio.

A functional level examination indicates that certain fields in a microcode exhibit an extremely biased ‘X’-bit distribution. The large immediate field, as a representative example, is either fully specified or fully unspecified. This field thus creates sizable variations in the number of X’s within a codeword, which in turn limits the attainable compression ratio of an XOR network. On the other hand, the immediate values can be effectively captured in a small LUT table, as a program typically does not utilize all the 2^k distinct immediate values provided by a k -bit immediate field.

If a set of control signals are always fully specified and exhibit a highly limited set of value combinations, it is also more desirable to capture them in a LUT table. Examples include *control-altering* signals, *interrupt* signals, as well as *write-enable* signals for register files and the memory. It is not desirable to compress these signals through an XOR network as they are always fully specified. On the other hand, these three groups of signals are *mutually exclusive* in that if any signal in one group is high, none of the signals in the other two groups can be high. This strict constraint in value combinations therefore enables these three groups of signals to be effectively captured by a small LUT table.

Except for the aforementioned two cases, the remaining set of control signals in a customized IP typically displays an appreciable amount of randomness in terms of both the ‘X’ bit distributions and the possible value combinations. Signals such as register names, due to its small width, would not create a sizable variation in the number of ‘X’ bits within a codeword. Meanwhile, as a program typically makes maximum utilization of the available registers, a large number of value combinations of register names will occur. Accordingly, it is more desirable to compress these signals using an XOR network than using a LUT table.

According to the examination outlined above, the decomposition procedure maximally identifies a set of columns that are either fully specified or fully unspecified, as well as a set of columns that are always fully specified yet exhibit limited value combinations. By capturing these two sets of control signals in a LUT table, the number of specified bits in the *hard-to-compress* codes can be sizably reduced, yet the number of ‘X’ bits is retained intact. The attainable compression ratio of the XOR network thus can be significantly improved. Meanwhile, as the LUT table only captures a small set of control signals with highly repetitive patterns, both the bitwidth and the number of entries in the LUT table are quite small.

At runtime, this small table and the XOR network are accessed in parallel using distinct fields of the compressed codes. Therefore, this hybrid compression is able to attain utmost code compression within a negligible amount of performance and hardware overhead.

4.4 Overall code compression flow

The overall code compression flow with all the three compression ratio enhancement techniques integrated is presented in Figure 7. The *column merging* procedure is first evoked to maximally reduce the width of the microcodes. Both the LUT width and the number of outputs of the XOR network can be reduced as a result. The remaining columns are subsequently decomposed into two sets, with the goal of maximally reducing the number of specified bits in the *hard-to-compress* codewords without sizably enlarging the LUT size. These two sets of columns are then compressed

Figure 7: Proposed hybrid code compression flow

individually, while for the XOR group the *column reordering* procedure is first evoked to attain a more random distribution of ‘X’ bits.

Although the column *reordering* and *merging* procedures have changed the original bit sequence of the control signals, this sequence still can be effectively restored at runtime. For embedded systems dedicated for a single application, this can be easily implemented through a customized routing of the interconnects without inducing any hardware overhead. On the other hand, for systems with more general applications, a fixed interconnect routing might not deliver the optimal compression for varying programs. In this case, a high-performance reconfigurable interconnect architecture [18, 19] can be incorporated in the decompression hardware to provide the routing flexibility required by different applications, at the cost of slightly increased area and performance overhead.

5. EXPERIMENTAL EVALUATION

To evaluate the proposed compression framework, three techniques have been implemented in our experimental studies: the standard LUT-based compression technique proposed in [13], the pure XOR network-based method, as well as the hybrid compression method. It has been reported in [13] that the LUT-based compression technique generally attains a minimum compression when the columns are divided into three sets with distinct dictionaries. Accordingly, this technique has been evaluated for both 1-dictionary and 3-dictionary configurations in our experimental studies. On the other hand, the number of dictionaries used in the hybrid compression approach is set to 2. To attain a fair comparison, the *column merging* technique proposed in Section 4.2 is consistently applied to all three compression techniques to maximally reduce the width of the microcodes.

The microcodes with unspecified values are generated using the NISC toolset [4], which provides a custom datapath and compiles a program described in a high-level language to an executable binary that directly drives the control signals of components in the datapath. In our experiments, the width of the microcode is 86 bits.

The most significant advantage of the proposed fixed-length compression technique is the extremely low on-chip storage requirement. The pure XOR network-based approach requires no dictionary, while the hybrid compression scheme only requires a small LUT. Accordingly, the three compression techniques are evaluated in terms of both the compression ratio and the LUT size.

Figure 8 presents the compression ratio attained by each technique. As can be seen, the pure XOR network-based

Figure 8: Compression ratio of various techniques

technique delivers a compression ratio comparable to that of the 1-dictionary LUT-based technique. On the other hand, the *hybrid* compression technique consistently delivers the lowest (the best) compression ratio among the four techniques for all the benchmarks. The average compression ratio attained by the hybrid technique is 0.326, a 18% improvement as compared to the average compression ratio of the 3-dictionary LUT-based technique. These results clearly confirm the efficacy of the proposed hybrid compression technique in combining the advantages of both the LUT-based and the XOR network-based techniques.

The first two sets of results in Table 2 present the size of the original microcoded program, as well as the LUT table size required by each technique. As can be seen, the LUT-based compression technique needs to capture 44% and 14% of the original code size in the on-chip LUT, respectively for the 1-dictionary and 3-dictionary configurations. This significant storage thus drastically degrades decompression speed. In contrast, the pure XOR network-based technique requires no on-chip storage at all, while the hybrid compression technique only needs to use 2.3% of the original code size in the on-chip LUT. This extremely low storage requirement thus reduces both the hardware cost and the leakage power consumption, while enabling the development of an extremely high speed decompressor as well.

The first two sets of results in Table 2 confirm that the LUT size required by a LUT-based compression technique is generally proportional to the original size of the program. Accordingly, for a microcoded IP that holds a set of applications, the required LUT size is usually determined by the application of the largest code size. In contrast, the LUT size required by the proposed hybrid compression technique is less sensitive to the original code size. As a result, even if the applications held by the microcoded IP display highly unbalanced code sizes, the on-chip LUT size can still be effectively controlled.

The decompression speed of the pure XOR network-based approach is extremely fast. As each output of the XOR network is produced in parallel, the hardware decompression only displays two levels of gate delay. Accordingly, the decompression speed of the hybrid compression technique is determined by the access latency of the on-chip LUT. We have employed Cacti [20] to evaluate the LUT access latency of the LUT-based compression and the hybrid compression techniques. The configuration of the largest LUT and the

Benchmark	Code size	LUT size				LUT configuration			Access latency		
		LUT-1D	LUT-3D	XOR	Hybrid	LUT-1D	LUT-3D	Hybrid	LUT-1D	LUT-3D	Hybrid
DCTU	21.164	12.466	4.604	0	0.485	185×69	119×23	34×9	1.537	1.318	0.950
Dijkstra	34.770	13.403	4.028	0	1.108	183×75	80×25	47×16	1.545	1.242	1.021
Mpeg	21.248	13.467	4.739	0	0.736	197×70	103×23	30×13	1.631	1.291	0.957
Qsort	77.182	26.163	7.675	0	1.109	367×73	167×24	53×13	1.650	1.326	1.024
Sha	49.719	24.640	7.319	0	1.325	332×76	133×25	67×16	1.647	1.377	0.976
Average	40.816	18.028	5.673	0	0.953	–	–	–	1.602	1.311	0.985

Table 2: LUT size (Kbits), the largest LUT and its access latency of each technique

corresponding access latency values are shown in the last two sets of results in Table 2. As can be seen, the LUT required in the hybrid compression technique exhibits both a smaller width and fewer entries as compared to the dictionaries used in the LUT-based compression technique. This in turn results in a 63% reduction in the overall access latency, achieved as a result of the reduced delay in both the address decoder and the output drivers.

Given the experimental results in compression ratio, LUT size, and access latency, it can be clearly concluded that the proposed hybrid compression method delivers utmost compression ratio as well as high speed decompression, achieved within a highly constrained amount of extra hardware.

6. CONCLUSIONS

We have proposed in this paper an extremely fast and cost-effective code compression technique for microcoded IPs. Through utilizing a linear network, the proposed technique can flexibly and precisely fill in the fully specified bits in each microcode. The linear property inherent in the compression strategy in turn enables the development of an extremely low-overhead decompression engine, composed of only a fixed-bandwidth XOR network. A set of functional level optimization approaches, including a *column reordering* and a *column merging* technique, have been proposed to further improve the compression ratio. Through combining the flexible XOR network with a minimum two-level storage for highly specified fields, such as immediate values, a *hybrid* compression technique is able to deliver utmost code compression within a negligible amount of storage overhead. Experimental results show that the proposed hybrid compression technique is able to attain an average compression ratio of 0.326, while only 2.3% of the original microcodes need to be stored in an on-chip LUT. Such high efficiency thus enables the incorporation of this compression technique into most microcoded IPs to attain utmost code size reduction within a negligible amount of performance and hardware overhead.

7. REFERENCES

- [1] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. R. Rau, D. Cronquist, and M. Sivaraman, "PICO-NPA: High-level synthesis of nonprogrammable hardware accelerators," *VLSI Signal Processing*, vol. 31, no. 2, pp. 127–142, 2002.
- [2] N. Clark, H. Zhong, K. Fan, S. Mahlke, K. Flautner, and K. V. Nieuwenhove, "OptimoDE: Programmable accelerator engines through retargetable customization," in *Hot Chips*, 2004.
- [3] S. Weber and K. Keutzer, "Using minimal minterms to represent programmability," in *CODES+ISSS*, Sept. 2005, pp. 63–68.
- [4] M. Reshadi, B. Gorjiara, and D. Gajski, "Utilizing horizontal and vertical parallelism with a no-instruction-set compiler for custom datapaths," in *ICCD*, Oct. 2005, pp. 69–76.
- [5] M. Thuresson, M. Sjalander, M. Bjork, L. Svensson, P. Larsson-Edefors, and P. Stenstrom, "FlexCore: Utilizing exposed datapath control for efficient computing," in *IC-SAMOS*, July 2007, pp. 18–25.
- [6] A. Wolfe and A. Chanin, "Executing compressed programs on an embedded RISC architecture," in *Microarchitecture*, Dec 1992, pp. 81–91.
- [7] T. M. Kemp, R. K. Montoye, J. D. Harper, J. D. Palmer, and D. J. Auerbach, "A decompression core for PowerPC," *IBM Journal of Research and Development*, vol. 42, no. 6, pp. 807–812, 1998.
- [8] K. D. Cooper and N. McIntosh, "Enhanced code compression for embedded risc processors," in *Programming Language Design and Implementation*, 1999.
- [9] S. K. Debray, W. Evans, R. Muth, and B. D. Sutter, "Compiler techniques for code compaction," *ACM Trans. on Programming Languages and Systems*, vol. 22, no. 2, 2000.
- [10] S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, thumb, and the ARM7TDMI," *IEEE Micro*, vol. 15, no. 5, pp. 22–30, 1995.
- [11] R. Grehan, "16-bit: The good, the bad, your options," in *Embedded Systems Programming*, vol. 12, no. 8, 1999.
- [12] G. G. Pechanek, S. Larin, and T. Conte, "Any-size instruction abbreviation technique for embedded DSPs," in *ASIC/SOC Conference*, Sept. 2002, pp. 8–12.
- [13] B. Gorjiara and D. Gajski, "FPGA-friendly code compression for horizontal microcoded custom IPs," in *FPGA'07*, 2007, pp. 108–115.
- [14] E. Borin, M. Breternitz, Y. Wu, and G. Araujo, "Clustering-based microcode compression," in *ICCD'07*, Oct. 2007, pp. 189–196.
- [15] M. Thuresson, M. Sjalander, and P. Stenstrom, "A flexible code compression scheme using partitioned look-up tables," in *HiPEAC*, Jan. 2009, pp. 95–109.
- [16] G. Stewart, *Introduction to Matrix Computations*. Academic Press, 1973.
- [17] I. Bayraktaroglu and A. Orailoglu, "The construction of optimal deterministic partitionings in scan-based BIST fault diagnosis: Mathematical foundations and cost-effective implementations," *IEEE Trans. Computers*, vol. 54, no. 1, pp. 61–75, 2005.
- [18] D. Kim, K. Lee, S.-J. Lee, and H.-J. Yoo, "A reconfigurable crossbar switch with adaptive bandwidth control for networks-on-chip," in *ISCAS*, Jan. 2005, pp. 2369–2372.
- [19] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. Rabaey, "Design methodology of a low-energy reconfigurable single-chip DSP system," *Journal of VLSI Signal Processing Systems*, vol. 28, pp. 47–61, 2001.
- [20] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," *Tech. report, HP Labs*, April 2008.