Single-ended Coding Techniques for Off-chip Interconnects to Commodity Memory

Mihir Choudhury, Kyle Ringgenberg, Scott Rixner, and Kartik Mohanram

Rice University

{mihir,kringg,rixner,kmram}@rice.edu

Abstract

This paper introduces a class of single-ended coding schemes to reduce off-chip interconnect energy consumption. State-of-the-art codes for processor-memory off-chip interfaces require the transmitter and receiver (memory controller and memory) to collaborate using current and previously transmitted values to encode and decode data. Modern embedded systems, however, cannot afford to use such double-ended codes that require specialized memories to participate in the code. In contrast, a single-ended code enables the memory controller to encode data stored in memory and subsequently decode that data when it is retrieved, allowing the use of commodity memories. In this paper, single-ended codes are presented that assign limited-weight codewords using trace-based mapping techniques. Simulation results show that such codes can reduce the energy consumption of an uncoded off-chip interconnect by up to 42.5%.

1. Introduction

Most modern system-on-a-chip-based (SoC-based) embedded systems require more memory capacity than can reasonably be embedded into the SoC core. In such systems, the interconnect between the core and external memory can consume as much or more energy than the core itself. Even though external memory and its associated interconnect are major contributors to the overall energy consumption in SoC-based embedded systems, such systems will continue to require the memory capacity afforded by external memory into the foreseeable future. Therefore, it is essential to develop memory controller architectures that reduce the energy consumption of the off-chip interconnect to external memory.

Many techniques have been proposed to reduce the energy consumption of off-chip interconnects and external memories. Energy can be reduced by minimizing the number of external memory accesses [1–4] and then using low energy memory modes when the external memory is not in use [5, 6]. However, external memory is included in most systems because that is the only way to provide the necessary memory capacity for the system. Therefore, a significant number of external memory accesses will always remain. Memory compression [7–11] and coding [12–25] are well-known techniques to further reduce the energy consumption of the processor-memory interconnect and the external memory. However, memory compression techniques are best suited for instructions, not data, which leaves coding as the primary technique for reducing energy consumption on the interconnect for data transfers.

Memory coding reduces energy consumption on the interconnect by reducing switching activity, i.e., bit transitions. However, prior

coding techniques are almost all context-dependent, double-ended codes that require a codec (coder/decoder) on both ends of the interconnect [12–24]. Context-dependent codes use a one-to-many mapping in which the codeword is chosen based upon both the current and previous data values to cross the interconnect. For example, bus-invert coding selects the value that minimizes the Hamming distance to the value on the interconnect, by either transferring the data value unchanged or inverting it [24]. Since the context of a transmission is only known at the time of the transfer, such context-dependent codes must also be double-ended, meaning that the receiver must participate in the code to recover the original data value.

Modern embedded systems, however, cannot afford to use double-ended codes over the processor-memory interconnect. As embedded systems development is driven by cost and time-to-market considerations, such systems must use commodity memories. These commodity memories do not have the logic required to participate in double-ended coding schemes. So, while double-ended coding schemes can reduce energy consumption when they are appropriate (such as for on-chip interconnects, specialized systems that can afford the use of custom memories, etc.), they are not useful for the vast majority of embedded systems.

This paper introduces the concept of single-ended, limited-weight codes for interconnect energy reduction. These single-ended codes can be used in modern embedded systems with commodity memory, yet are still able to achieve significant reduction in the processor-memory interconnect's energy consumption. The proposed codes separate codeword generation from codeword assignment. The generated codewords are limited-weight codewords (codewords with a limited number of ones), which are selected from a larger code space [25]. As initially proposed, such limited-weight codes (LWCs) were used with a straightforward assignment strategy to implement a single-ended, context-independent code. However, using such an assignment, LWCs only modestly reduce the switching activity on the interconnect compared to state-of-the-art codes. This paper introduces a trace-based assignment strategy for limited-weight codewords that reduces the switching activity on the interconnect as much or more than state-of-the-art double-ended, context-dependent codes without requiring the participation of the memory.

The trace-based assignment strategy exploits frequency and sequence information from memory traces collected on the target system. Previously proposed frequency-based codes have all been double-ended [12–16]. A key observation of this work is that the appropriate use of frequency information can result in significant reductions in switching activity on the interconnect without the need to resort to a double-ended coding scheme. Furthermore, the additional use of sequence information during codeword as-

signment can lead to even greater reductions in switching activity. In fact, a sequence-based assignment of limited-weight codewords yields a 42.5% reduction in switching activity with a single-ended code. This is competitive with one of the best previously proposed double-ended, frequency-based codes, which achieves a 38.7% reduction in switching activity with DRAM participation.

This paper is organized as follows. In Section 2, the proposed singled-ended, limited-weight codes are described. Section 3 describes a memory controller architecture to support these coding techniques. Section 4 analyzes the performance of the proposed memory controller innovations on a set of embedded computing benchmarks. Finally, Section 5 concludes the paper.

2. Coding for Commodity Memories

This section develops a class of context-independent, single-ended coding schemes for embedded systems. These coding schemes are split into two phases. During the first phase, a set of codewords is generated. In the second phase, each information symbol is assigned to a unique codeword. These assignments are determined using only frequency-based and sequence-based metrics, without any run-time context information. Such codes have several advantages. First, they significantly lower energy consumption on the interconnect between the SoC and the memory modules. Second, they are single-ended, so they do not require the memory to participate in the coding/decoding process. A codec is only required in the memory controller on the SoC. Last, the coding/decoding process has a negligible impact on performance.

2.1 Codeword Generation

Limited-weight codes (LWCs) provide an effective set of codewords that minimize the number of ones in each codeword [25]. LWCs are single-ended, context-independent codes. Consider a k-bit wide data bus with 2^k information symbols. A m-LWC is a one-to-one mapping where every word in the 2^k input space maps to a codeword such that the Hamming weight (i.e., the number of ones in the codeword) is less than or equal to m. Since the source entropy must remain unchanged, i.e., since every information symbol must have a unique codeword, the following inequality must be satisfied by all m-LWCs:

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{m} \geq 2^k \qquad (1)$$

Here, n is the minimum number of bits ($m \leq n$) required to satisfy the inequality (1). Therefore, n determines the width of the bus needed to implement a m-LWC. Note that (1) is only satisfied for $n \geq k$.

A perfect m-LWC satisfies (1) above with equality, i.e., all the codewords of length n with weight less than or equal to m are used in the mapping. For example, a 4-LWC where k equals 8 is a perfect 4-LWC when the codeword bus width, n, equals 9. The degenerate case where m equals k is a simple remapping. For example, an 8-LWC where k equals 8 includes the same codewords as the input space, but they can be potentially reassigned to reduce energy consumption on the processor-memory interconnect. This paper focuses on 4-LWC and 8-LWC codewords for 8-bit information symbols.

2.2 Trace-based Assignment

As discussed in the introduction, most coding techniques use context information to assign codewords in order to minimize transitions. However, single-ended codes require the assignment to be independent of the current context. This can be done with no *a pri-*
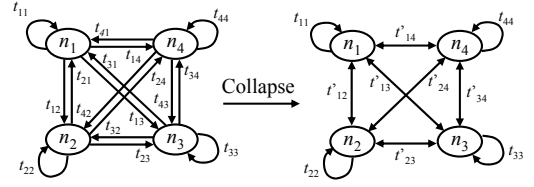

Figure 1: Graph representation of trace information

ori information about the memory behavior of the system, as in the originally proposed limited-weight code [25]. In contrast, this paper proposes to assign codewords based on an analysis of memory traces taken from the system.

Two important sets of information can be collected from a memory trace: the frequency of occurrence of each value, i, that crosses the interconnect (f_i) and the frequency with which each value, i, that crosses the interconnect is followed by every other value, j (t_{ij}). This information can be represented by a graph, as shown in Figure 1. Each node in the graph represents a single data value, i. The weight of i is the frequency of occurrence of that data value, f_i. Each node, i, is also connected to every other node, j, including itself, with a directed edge. The weight of an edge from node i to node j is the the frequency with which the data value j follows the data value i over the interconnect, t_{ij}. Since the same number of bit transitions will occur regardless of the order in which two values cross the interconnect, the two directed edges that connect each pair of nodes can be collapsed into a single edge whose weight is the sum of the two directed edges, t'_{ij}:

$$t'_{ij} = t'_{ji} = t_{ij} + t_{ji} \quad \forall i \neq j \qquad (2)$$

Note that the frequency of occurrence of each value is then related to the transition frequencies in the following way:

$$f_i = \sum_j t_{ij} = t_{ii} + \frac{1}{2} \sum_{j \neq i} t'_{ij} \qquad (3)$$

Note that $\sum_{j \neq i} t'_{ij}$ is halved because t'_{ij} includes all the edges that both enter and leave node i, and each occurrence of i is accompanied by both an entrance and an exit from i.

This graph can then be used to assign codewords in a variety of ways. This paper explores the use of the node weights to perform frequency-based assignment and the edge weights to perform sequence-based assignment.

2.2.1 Frequency-based Assignment

Codewords can be assigned based on the frequency of occurrence of each information symbol. Several previously proposed codes have performed such frequency-based assignment [12–16]. However, they have used different code generation techniques. For example, frequent value coding uses one-hot encoding of the 32 most frequently occurring 32-bit values [15]. Since most of the values remain uncoded, frequent value coding also uses a decorrelator [17, 20] to reduce switching activity using context information. Single-ended versions (without the decorrelator) of such word-level frequent value coding schemes do not perform as well as simpler coding schemes, such as bus-invert coding [24].

Fig. 2 presents the frequency of occurrence of word and byte values across the MiBench embedded benchmark suite [26]. This data helps illustrate the limitations and opportunities of frequency-based coding techniques. As Fig. 2(a) shows, the top 3 32-bit words occur orders of magnitude more frequently than other values on the processor-memory interconnect. However, as the figure also shows, the remaining values in the top 256 frequently occurring
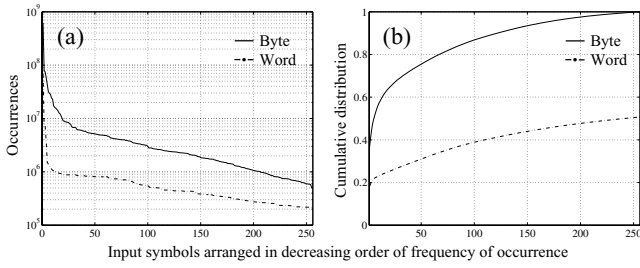
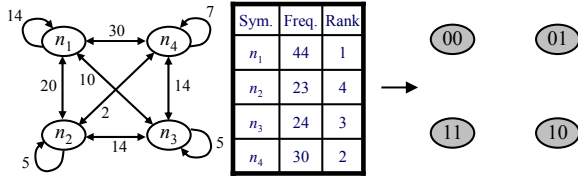**Figure 2: Frequency of the top 256 Byte and Word Values**

Figure 3: Frequency-based assignment

words continue to have a relatively high frequency of occurrence that does not drop off dramatically. For instance, the 10th most frequently occurring value only occurs only 5 times as often as the 256th most frequently occurring value. These remaining words are likely to be significant contributors to the switching activity on the interconnect. This is further illustrated in Fig. 2(b), where the cumulative distribution of the top 256 most frequently occurring 32-bits words only accounts for approximately 50% of the values on the processor-memory interconnect. Given that so many different word values cross the interconnect, it is impractical to encode them all efficiently. For example, Fig. 2(b) shows that frequent-value coding would only encode 27.6% of the interconnect traffic by one-hot encoding the 32 most frequently occurring values.

It is far more practical to code at the byte level, since there are only 256 possible values. Furthermore, similar to the word-level, the most frequently occurring byte appears an order of magnitude more frequently than other bytes on the processor-memory interconnect. Although the drop-off in frequency is a little less rapid than at the word-level, the 200 least frequently occurring byte values still have surprisingly similar frequencies of occurrence. The relatively slow drop-off in frequency of occurrence suggests that a code that remaps *all* information symbols to limited-weight codewords will be effective.

A frequency-based, limited-weight code exploits this frequency distribution by ordering the information symbols by decreasing frequency of occurrence, f_i. Each information symbol is then assigned, in order, to the LWC codeword with the least weight that remains. The use of an 8-LWC allows a simple remapping in which information symbols are reassigned based on their frequency of occurrence. The use of a 4-LWC can be more effective, as the weight of the codewords is reduced. Such a 4-LWC would map the 46 $\left(\binom{9}{0} + \binom{9}{1} + \binom{9}{2}\right)$ most frequently occurring byte values (which account for 74% of all bytes transferred over the interconnect) to 9-bit codewords with 2 or fewer ones in the codeword. The remaining 26% of the bytes transferred over the interconnect would be mapped to 9-bit codewords with only 3 or 4 ones in the codeword. Such a code will result in a dramatic decrease in ones transferred over the interconnect, which will increase the probability of transferring consecutive zeros across any given wire. Therefore, interconnect values will tend to remain at zero, reducing the overall switching activity.

Frequency-based remapping is illustrated with an example shown in Fig. 3. There are 4 nodes in the graph and the code-space consists of all 2-bit codewords, i.e., a 2-LWC where $n = 2$. The second column of the table in the figure shows the frequency of occurrence, f_i, of each value, and the third column ranks the nodes in descending order of f_i. Based upon this, the nodes are processed in the order n_1, n_4, n_3, and n_2 and the assigned codewords are illustrated in the resulting graph with shaded nodes. The expected number of bit transitions for this assignment of codewords, if the relative frequency of occurrence remains unchanged, is 124.

2.2.2 Sequence-based Assignment

The set of codewords can also be assigned to information symbols based on the sequence in which they occur. Specifically, the frequency with which pairs of values follow each other on the interconnect can be used to ensure that pairs of values that follow each other frequently will be assigned codewords that are close to each other (i.e., have a small Hamming distance between them). Similarly, pairs of values that do not follow each other very frequently can be assigned codewords that are not close to each other.

Figure 4 shows the frequency of transitions between bytes as a function of the frequency of occurrence of the bytes. The 256 8-bit information symbols are arranged in 8 groups of 32 symbols each in decreasing order of frequency of occurrence. For example, group 1 contains the 32 most frequently occurring bytes and group 8 contains the 32 least frequently occurring bytes. Along the y-axis, the vertically stacked bars represent the fraction of the transitions that occur between a symbol in group i and a symbol in group j, where $j \geq i$. From the figure, it is clear that approximately 38% of the transitions are completely contained within group 1 of the 32 most frequently occurring bytes. However, it is surprising that 45% of the transitions also occur between bytes in group 1 and bytes in the remaining groups. Indeed, the number of transitions that occur between a byte in group 1 and a byte in group 2 (second bar from the bottom in the group 1 bar) is nearly an order of magnitude greater than transitions that are completely contained between bytes in group 2 (bottom bar in the group 2 bar). This is a very strong observation that motivates codeword assignments for bytes in groups 2 through 8 that reduce the Hamming distance to the codewords assigned to bytes in group 1. This is, however, not addressed in frequency-based assignment, since the codewords are ranked in ascending order of their weights and assigned to information symbols ranked in descending order of frequency of occurrence. As a result, frequency-based assignment minimizes the Hamming distance between codewords assigned to bytes in a group, at the expense of the Hamming distance between codewords assigned to bytes across groups. The rest of this section describes sequence-based codeword assignment that utilizes transition frequency distribution between pairs of bytes across groups to minimize the Hamming distance of their codeword assignments.

From the graph in Fig. 1, sequence-based assignment is formally equivalent to the minimization of the following objective function:

$$\sum_{i=1}^{256} \sum_{j=i+1}^{256} \left(\text{code}[i] \oplus \text{code}[j]\right) t'_{ij} \qquad (4)$$

where $\text{code}[i]$ and $\text{code}[j]$ are the codewords assigned to information symbols i and j, respectively, and t'_{ij} is the frequency of occurrence of the sequences $i \rightarrow j$ and $j \rightarrow i$, as defined in equation (2). This problem is intractable and belongs to the class of $\mathcal{NP}$-hard problems.

In practice, however, such problems respond fairly well to heuristics that proceed with assignments to information symbols
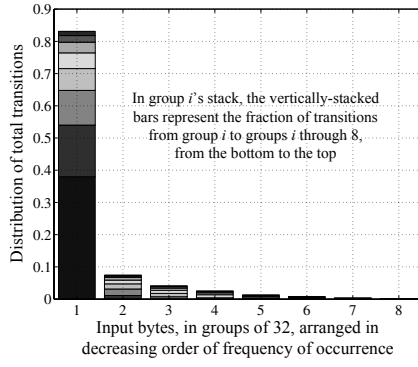
Figure 4: Frequency of sequences between groups of bytes sorted by frequency of occurrence

$\mathcal{C}$ – set of generated codewords, e.g., 8-LWC, 4-LWC for assignment
$\mathcal{U}$ – set of information symbols with no codeword assignment
$\mathcal{A}$ – set of information symbols with assigned codewords

initialize
$\mathcal{U} \leftarrow$ {all nodes in subject graph}
$n \leftarrow \max_{i \in \mathcal{U}}(f_i)$, i.e., n is the most frequent node
$\text{code}[n] \leftarrow 0; \ \mathcal{C} \leftarrow \mathcal{C}\backslash\{0\}; \ \mathcal{U} \leftarrow \mathcal{U}\backslash\{n\}; \ \mathcal{A} \leftarrow \{n\}$

while $\mathcal{U} \neq \phi$ **do**
$n \leftarrow \max_{i \in \mathcal{U}}\left(\sum_{j \in \mathcal{A}} t'_{ij}\right)$
$\text{code}[n] \leftarrow \min_{c \in \mathcal{C}}\left(\sum_{j \in \mathcal{A}}(c \oplus \text{code}[j]) \, t'_{nj}\right)$
$\mathcal{C} \leftarrow \mathcal{C}\backslash\{\text{code}[n]\}; \ \mathcal{U} \leftarrow \mathcal{U}\backslash\{n\}; \ \mathcal{A} \leftarrow \mathcal{A}\bigcup\{n\}$

Figure 5: Pseudo-code for sequence-based assignment

one-at-a-time. The linear pass heuristic algorithm shown in Fig. 5 was developed to solve this problem as follows. The central idea is to proceed by making codeword assignments to nodes (information symbols) one-at-a-time in the subject graph. All nodes begin in the set of unassigned nodes, $\mathcal{U}$. Once a codeword assignment is determined for a node, that node is transferred to the set of assigned nodes, $\mathcal{A}$. The first symbol to be assigned a codeword is the most frequently occurring byte. Although the first codeword can be randomly assigned, 0 is chosen for simplicity. On each subsequent pass, the node, n, with the maximum sum of transition frequencies to $\mathcal{A}$ given by $\sum_{j \in \mathcal{A}} t'_{nj}$ is chosen for codeword assignment. Ties are broken by selecting the n with higher frequency of occurrence, f_n. The codeword, c, is chosen such that it has the minimum cumulative weighted Hamming distance to the nodes in $\mathcal{A}$. The weighted Hamming distance of the codeword c to node $j \in \mathcal{A}$ is the Hamming distance between c and $\text{code}[j]$ weighted by the corresponding t'_{nj}. By summing over all $j \in \mathcal{A}$, the cumulative weighted Hamming distance of c to $\mathcal{A}$ is determined. The search for the codeword that minimizes the cumulative weighted Hamming distance is exhaustive over the pool of unassigned codewords. Once a codeword is assigned to n, it is moved from $\mathcal{U}$ to $\mathcal{A}$. The updates are performed as indicated in the pseudo-code, and a new node, n, is chosen for codeword assignment on the next pass.

The heuristic is illustrated with an example shown in Fig. 6, starting with the same graph as Fig. 3. There are 4 nodes in the graph and the code-space consists of all 2-bit codewords. Each graph in the figure represents one pass through the algorithm of Fig. 5. The shaded nodes belong to the set of assigned nodes, $\mathcal{A}$, and the unshaded nodes belong to the set of unassigned nodes, $\mathcal{U}$. At each step, the solid edges are edges that are considered in selecting the next node, n, and the dashed edges are ignored at that step, but will
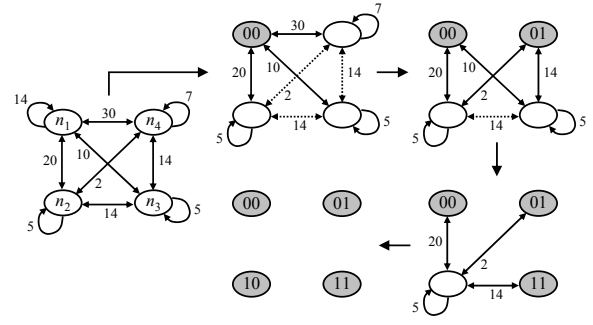

Figure 6: Example for sequence-based assignment

be used in future steps. For clarity, edges that are no longer needed are removed from each graph. The set of assigned nodes, $\mathcal{A}$, is initialized to the most frequently occurring node n_1, which is assigned the codeword 00, as shown in the upper left of Fig. 6. The weights of the edges between the remaining nodes and $\mathcal{A}$ are used to select node n_4, which has the maximum weight to $\mathcal{A}$. This node is assigned a codeword that minimizes its cumulative weighted Hamming distance to $\mathcal{A}$, as shown in the upper right of Fig. 6. The process is repeated with the new $\mathcal{A}$, which now contains two nodes. Node n_2 has edge weights of 22 to $\mathcal{A}$ and node n_3 has edge weights of 24 to $\mathcal{A}$, so node n_3 is assigned a codeword next. As shown in the lower right of Fig. 6, node n_3 is assigned 11, as that minimizes the cumulative weighted sum of the bit transitions between n_3 and nodes $\{n_1, n_4\}$ in $\mathcal{A}$. Finally, the last node n_2 is assigned the remaining codeword, as shown in the lower left of the figure. The expected switching activity obtained for this assignment of codewords is 102. This is a 17.7% improvement over the frequency-based assignment for the same graph, and is attributable to the interchange of the assignments of 10 and 11 to nodes n_2 and n_3 in the subject graph. Sequence-based assignment thus results in the best possible assignment, since edges between the two pairs of codewords with Hamming distance 2 ($\{01,10\}$ and $\{00,11\}$) have the two lowest weights (2 and 10) in the subject graph.

3. Memory Controller Architecture

A context-independent codec does not need to be near the pins of the SoC core. As the only input to the codec is the data itself, the data can be encoded and decoded anywhere within the memory controller. Therefore, data can be encoded well before it is written to memory. The latency of encoding the data can likely be hidden by the long latency SDRAM operations that must occur before the data can cross the pins. Similarly, data can be decoded well after it is read from memory. Again, the decoding latency could possibly be hidden by arbitration delays for the system bus. For the 30 benchmark programs from the MiBench suite that will be explored in Section 4, an extra cycle latency penalty for decoding results in less than a 0.1% performance penalty on average.

Of the many possible ways to implement a context-independent codec, a lookup-table is the most efficient mechanism for the trace-based codes described in this paper. To encode or decode bytes, a 256-entry table would be required with either 8 or 9 bit entries, depending on the code. For performance, multiple identical tables could be used, one for each byte that can be transferred on the off-chip interconnect in a given cycle. To provide the flexibility to change the code, the lookup-tables would have to be SRAM structures.

The introduction of SRAM tables in the encoder and decoder will increase the total energy consumed by the system. For the frequency-based, 4-LWC code, the encoder would require four 256

entry, 9-bit wide SRAM tables to allow four bytes to be encoded per cycle. The decoder would require four 512 entry, 8-bit wide SRAM tables to allow four bytes to be decoded per cycle. Note that half of the 512 entries in the decoder tables would be unused, but a 512 entry SRAM enables a simple lookup for decoding. These tables can be approximated as 2 KB, 32-bit wide memories. An access to such a memory in 65 nm CMOS at 0.9 V would consume approximately 5.7 pJ [27].

To put this in perspective, a transition on a 5 mm processor-memory interconnect with a 2.5 V memory would consume approximately 11 pJ. This assumes that the capacitance of the processor-memory interconnect (including the processor package traces, processor pins, board trace, memory pins, and memory package traces) is about 1.5 pF for the processor, 106 pF/m for the board trace, and 1.5 pF for the memory [28]. Therefore, the proposed coding schemes must eliminate about $1/2$ a bit transition per transfer to break even in terms of energy consumption.

Finally, many of the codes discussed here increase the size of the data by adding an additional bit for every byte. This increases the data-path width of the memory controller, the width of the processor-memory interconnect, and the width of the SDRAM. Obviously, this additional bit can increase energy consumption. However, the objective of these codes is to reduce energy consumption by limiting the number of transitions, so usually this is not an issue in the memory controller or the processor-memory interconnect, as will be shown in Section 4 (all results include the transitions on this additional wire, as appropriate). However, widening the SDRAM is potentially problematic. Samsung has introduced SDRAMs with 9-bit bytes, which consume 6–8% more current than their normal counterparts [29]. However, this is assuming a regular data pattern. In practice, the reduction in switching activity achieved by these codes can more than offset this increase.

4. Results

The coding techniques presented here were evaluated using the SimpleScalar/ARM simulator [30]. The simulator was configured to closely match the Intel Xscale processor [31]. SimpleScalar was also modified to incorporate a cycle accurate SDRAM model that simulates all timing and resource constraints [32]. The simulator is configured to model a 75 MHz, 512 Mb Micron MT48LC32M16A2-75 single data rate SDRAM. The switching activity on the interconnect for the coded and uncoded data transfers was calculated as the SDRAM is accessed, ensuring the bit transitions occur on the data bus in the same order as a real system.

The MiBench embedded benchmark suite was used to evaluate the proposed codes [26]. Thirty applications, spanning the automotive, consumer, networking, office, security, and telecommunication domains, are used from the suite with their large input sets.

Table 1 shows the average reduction in switching activity on the processor-memory interconnect for thirteen coding strategies when compared with the baseline uncoded case. The table shows the bus width for each coding strategy (switching activity on any additional wires are accounted for in all results), the average switching activity per transfer, and the reduction in switching activity compared to the uncoded case. The first two codes in the table are context-dependent, double-ended codes. Bus-invert coding is the simplest and most popular such code [24]. FV32 with a decorrelator one-hot codes the 32 most frequently occurring values (for each benchmark) and uses a decorrelator to significantly reduce switching activity [15]. As the table shows, both context-dependent, double-ended codes perform quite well, reducing switching activity on the interconnect by 21.8% and 38.7%, respectively.

The remaining eleven codes are all context-independent, single-ended codes that can be implemented entirely within the memory controller without specialized SDRAM. FV32 and FV8 are modified from the codes presented in [15] to make them single-ended. They simply one-hot encode the 32 most frequently occurring word values or the eight most frequently occurring byte values to form a code. These codes are labeled "Self", as each benchmark uses the most frequently occurring values from that benchmark. As the table shows, these codes yield only a 17.8% and 15.5% reduction in switching activity. Therefore, such a one-hot encoding strategy relies heavily on a context-dependent, double-ended decorrelator to reduce switching activity on the interconnect.

4-LWC is the original limited-weight code, presented in Section 2.1, which uses nine bits per byte to code all byte values with at most four bits set [25]. Without using trace-based assignment, 4-LWC is only able to reduce switching activity by 13.9%.

The next four limited-weight codes use the frequency-based assignment scheme presented in this paper. "Self" and "Global" refer to whether each benchmark's own frequency distributions were used to assign codewords for that benchmark or all benchmarks used the same codewords derived from the combined frequency distributions of all benchmarks. The 8-LWC is equivalent to remapping, and uses eight bits per byte. The 4-LWC uses nine bits per byte. As the table shows, these codes are able to reduce switching activity on the interconnect by 22.4–30.3% on average. As would be expected, the codes that use the frequency distributions for each benchmark individually yield about 5–6% higher reductions.

The last four limited-weight codes use the sequence-based assignment scheme presented in this paper. Similar to frequency-based coding, "Self" and "Global" refer to the use of self or global transition frequency distributions for codeword assignment. The codewords for the 8-LWC and 4-LWC schemes are equivalent to those for the frequency-based mapping technique. As the table shows, these codes are able to reduce switching activity on the interconnect by 33.3–42.5% on average. As would be expected, the codes which use the transition frequency distributions for each benchmark individually yield higher reductions, by about 7–8%.

The results show that when using limited-weight codes, the assignment strategy is critical. Furthermore, the penalty of using an extra wire per byte for the 4-LWC codes is more than offset by the effectiveness of such codes. Finally, these codes reduce the number of bit transitions per transfer by 2.7–5.1, indicating that the energy savings achieved by eliminating this many bit transitions per transfer using the proposed techniques dwarfs the minor energy overhead (~ 0.5 bit transitions per transfer) of implementing the SRAM-based lookup tables to support such codes in hardware.

5. Conclusions

State-of-the-art coding techniques to reduce energy consumption across the processor-memory interconnect have traditionally used double-ended techniques that require specialized memories. In contrast, the proposed trace-based, limited-weight codes are viable single-ended codes that are not only competitive with these state-of-the-art codes, but also compatible for use with commodity memory.

This paper has shown that both the type of code used and the assignment scheme for that code are important. Limited-weight codes by themselves are ineffective. Similarly, using frequency information without limited-weight codes yields an inefficient code that is also ineffective. However, using frequency or sequence information to assign limited-weight codes minimizes switching activity to a greater extent than any other context-independent, single-ended code. Furthermore, such codes sometimes outperform context-dependent, double-ended codes that cannot be used with

Table 1: Average reduction in switching activity

Code			Bus Width	Bit Transitions per Transfer	Reduction (%)
Uncoded			32	11.95	—
Context-dependent Double-ended	Prior work	Bus Invert ([24])	36	9.35	21.8
		Self FV32 with Decorrelator ([15])	33	7.33	38.7
Context-independent Single-ended	Prior work	Self FV32 (modified from [15])	33	9.82	17.8
		Self FV8 (modified from [15])	36	10.10	15.5
		4-LWC ([25])	36	10.29	13.9
	Frequency-based Assignment	Self 8-LWC	32	8.58	28.2
		Global 8-LWC	32	9.27	22.4
		Self 4-LWC	36	8.32	30.3
		Global 4-LWC	36	8.94	25.1
	Sequence-based Assignment	Self 8-LWC	32	6.99	41.5
		Global 8-LWC	32	7.97	33.3
		Self 4-LWC	36	6.87	42.5
		Global 4-LWC	36	7.71	35.5

commodity SDRAMs. Since embedded systems continue to use commodity memories and since the processor-memory interconnect is a dominant consumer of energy in such systems, the coding techniques presented here can significantly improve the overall energy efficiency of modern embedded systems.

6. References

[1] F. Catthoor *et al.*, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[2] C. Kulkarni, F. Catthoor, and H. DeMan, "Code transformations for low power caching in embedded multimedia processors," in *Proc. Intl. Parallel Processing Symposium*, 1998.

[3] C. Kulkarni *et al.*, "Cache conscious data layout organization for embedded multimedia applications," in *Proc. Design Automation and Test in Europe Conference*, 2001.

[4] P. R. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 3, 2000.

[5] V. Delaluz *et al.*, "DRAM energy management using software and hardware directed power mode control," in *Proc. Intl. Symposium on High-Performance Computer Architecture*, 2001.

[6] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for DRAM power management," in *Proc. Intl. Symposium on Low Power Electronics and Design*, 2001.

[7] S. Liao, S. Devadas, and K. Keutzer, "Code density optimization for embedded DSP processors using data compression techniques," in *Proc. Conference on Advanced Research in VLSI*, 1995.

[8] A. Wolfe and A. Chanin, "Executing compressed programs on an embedded risc architecture," in *Proc. Intl. Symposium on Microarchitecture*, 1992.

[9] L. Benini *et al.*, "Selective instruction compression for memory energy reduction in embedded systems," in *Proc. Intl. Symnposium on Low Power Electronics and Design*, 1999.

[10] H. Lekatsas *et al.*, "Code compression for low power embedded systems design," in *Proc. Design Automation Conference*, 2000.

[11] L. Benini *et al.*, "Hardware-assisted data compression for energy minimization in systems with embedded processors," in *Proc. Design, Automation and Test in Europe Conference*, 2002.

[12] K. Basu *et al.*, "Power protocol: reducing power dissipation on off-chip data buses," in *Proc. Intl. Symposium on Microarchitecture*, 2002.

[13] V. Wen *et al.*, "Exploiting prediction to reduce power on buses," in *Proc. Intl. Symposium on High-Performance Computer Architecture*, 2004.

[14] J. Yang and R. Gupta, "Frequent value locality and its applications," *ACM Transactions on Embedded Computing Systems*, vol. 1, no. 1, 2002.

[15] J. Yang, R. Gupta, and C. Zhang, "Frequent value encoding for low power data buses," *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 3, 2004.

[16] D. C. Suresh *et al.*, "A tunable bus encoder for off-chip data buses," in *Intl. Symposium Low Power Electronics and Design*, 2005.

[17] L. Benini *et al.*, "Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems," in *Proc. Great Lakes Symposium on VLSI*, 1997.

[18] L. Benini *et al.*, "Architectures and synthesis algorithms for power-efficient bus interfaces," *IEEE Trans. Computer-aided Design*, vol. 19, no. 9, 2000.

[19] H. Deogun *et al.*, "Leakage- and crosstalk-aware bus encoding for total power reduction," in *Proc. Design Automation Conference*, June 2004.

[20] E. Musoll *et al.*, "Exploiting locality of memory references to reduce the address bus energy," in *Proc. Intl. Symposium on Low Power Electronics Design*, 1997.

[21] S. Ramprasad, N. R. Shanbag, and I. N. Hajj, "A coding framework for low power address and data buses," *IEEE Transactions on VLSI Systems*, vol. 7, June 1999.

[22] P. Sotiriadis and A. Chandrakasan, "Low power bus coding techniques considering inter-wire capacitances," in *Proc. Custom Integrated Circuits Conference*, 2000.

[23] P. Sotiriadis, V. Tarokh, and A. P. Chandrakasan, "Energy reduction in VLSI computation modules: An information-theoretic approach," *IEEE Trans. Information Theory*, vol. 49, no. 4, 2003.

[24] M. R. Stan and W. P. Burleson, "Bus invert coding for low power I/O," *IEEE Transactions on VLSI Systems*, vol. 3, no. 1, 1995.

[25] M. R. Stan and W. P. Burleson, "Low-power encodings for global communication in CMOS VLSI," *IEEE Transactions on VLSI Systems*, vol. 5, no. 4, 1997.

[26] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE Workshop on Workload Characterization*, 2001.

[27] M. Q. Do *et al.*, "Parameterizable architecture-level SRAM power model using circuit-simulation backend for leakage calibration," in *Proc. Intl. Symposium Quality Electronic Design*, 2006.

[28] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge University Press, 1998.

[29] Samsung, "256/288 Mbit RDRAM K4R571669D/K4R881869D data sheet, version 1.4," July 2002.

[30] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, Feb. 2002.

[31] L. T. Clark *et al.*, "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE Journal of Solid-state Circuits*, vol. 36, no. 11, 2001.

[32] S. Rixner, "Memory controller optimizations for web servers," in *Proc. Intl. Symposium on Microarchitecture*, 2004.