Commutative Data Reordering: A New Technique to Reduce Data Movement Energy on Sparse Inference Workloads

Ben Feinberg*, Benjamin C. Heyman†, Darya Mikhailenko†, Ryan Wong†, An C. Ho†, and Engin Ipek†‡
*Sandia National Laboratories
Albuquerque, NM 87185, USA
†Department of Electrical and Computer Engineering, ‡Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
*bfeinbe@sandia.gov, †{bheyman, dmikhail, rwong5, aho7}@ece.rochester.edu ‡ ipek@cs.rochester.edu

***Abstract*—Data movement is a significant and growing consumer of energy in modern systems, from specialized low-power accelerators to GPUs with power budgets in the hundreds of Watts. Given the importance of the problem, prior work has proposed designing interconnects on which the energy cost of transmitting a 0 is significantly lower than that of transmitting a 1. With such an interconnect, data movement energy is reduced by encoding the transmitted data such that the number of 1s is minimized. Although promising, these data encoding proposals do not take full advantage of application level semantics. As an example of a neglected optimization opportunity, consider the case of a dot product computation as part of a neural network inference task. The order in which the neural network weights are fetched and processed does not affect correctness, and can be optimized to further reduce data movement energy.**

This paper presents commutative data reordering (CDR), a hardware-software approach that leverages the commutative property in linear algebra to strategically select the order in which weight matrix coefficients are fetched from memory. To find a low-energy transmission order, weight ordering is modeled as an instance of one of two well-studied problems, the Traveling Salesman Problem and the Capacitated Vehicle Routing Problem. This reduction makes it possible to leverage the vast body of work on efficient approximation methods to find a good transmission order. CDR exploits the indirection inherent to sparse matrix formats such that no additional metadata is required to specify the selected order. The hardware modifications required to support CDR are minimal, and incur an area penalty of less than 0.01% when implemented on top of a mobile-class GPU. When applied to 7 neural network inference tasks running on a GPU-based system, CDR respectively reduces average DRAM IO energy by 53.1% and 22.2% over the data bus invert encoding scheme used by LPDDR4, and the recently proposed Base+XOR encoding. These savings are attained with no changes to the mobile system software and no runtime performance penalty.

***Index Terms*—Memory Architecture, Optimization**

Supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

I. Introduction

Over the past decade, moving and marshalling data has eclipsed computation as the primary consumer of energy on many workloads. One recent analysis found that for a double precision fused multiply-add (FMA) operation, moving the operands from the edge of the IC to its center consumes $10\times$ the energy of the operation itself, while fetching the operands from off-chip memory consumes an additional $50\times$ [17]. Recent scaling trends of the off-chip memory show that bandwidth is increasing faster than cost per bit is decreasing, thereby increasing the peak power consumption of off-chip memories [32]. The high cost of data movement is especially acute when dealing with the large weight matrices that power modern machine learning workloads. Even with aggressive model compression techniques that allow the weight matrices to fit in on-chip SRAM, the energy consumed by reading the weights from large SRAM arrays represents a substantial fraction of the total system energy [23].

To help mitigate these concerns, researchers have proposed numerous energy-efficient data encoding schemes. These proposals rely on designing the interconnects to exhibit asymmetries in data transmission costs—*i.e.,* transmitting a **0** is significantly less costly than transmitting a **1**—and encoding the data to reduce the number of **1**s. Several successful encoding techniques exploit data similarity, representing the data based on the bitwise XOR between current and previous transmissions [32], [42], [50]. Although these proposals can save significant energy, they suffer from an important shortcoming: the encoded data is treated as a mere stream of bits, ignoring the semantics of the application within which the data are processed. Considering application level semantics could allow the encoding scheme to be specialized, unlocking new optimization opportunities.

This paper presents commutative data reordering (CDR), a hardware-software approach that reduces data movement energy on the linear algebra kernels at the heart of neural network inference tasks. CDR leverages the commutative

property of FMA operations to fetch the data from memory in an order that minimizes the energy. In these inference tasks, the same weight matrix is used repeatedly; for example, a neural network for eye tracking as part of an augmented reality application may access the learned weight matrix millions of times [30], and often on devices with significantly lower power budgets than the systems used to train the models [51]. CDR leverages this observation to reorder the weight matrix coefficients offline, and amortizes the overhead over many accesses at runtime.

CDR is applied to neural network inference tasks, which commonly use sparse weight matrices to significantly reduce storage costs. CDR exploits the fact that these sparse matrices store the data in <coordinate, value> tuples, such that matrix coefficients can be freely reordered with no additional metadata. The operand reordering problem solved by CDR is modeled as an instance of the well-studied Traveling Salesman [33] and Capacitated Vehicle Routing problems, allowing us to leverage the vast body of research on constrained optimization to explore the space of possible reorderings.

A naïve approach to reordering can cause significant performance penalties as it disrupts the spatial locality in the vector access stream. To remedy this problem, CDR uses *strided reordering*, wherein restrictions are placed on which coefficients can be reordered. Strided reordering provides a mechanism to trade-off the energy benefits of unrestricted reordering against greater performance via a tunable stride length parameter.

By leveraging application level semantics, CDR provides an average of 22.2% reduction in DRAM IO energy, and up to 30.6% as compared to a state-of-the-art data encoding proposal [32]. **These energy savings are attained with no increase in execution time, no changes to client software, and no area overhead beyond the previously proposed Base+XOR encoder. The low overhead of these changes makes CDR a low risk optimization on energy-constrained platforms.**

II. Background

CDR builds upon prior work on energy efficient interconnects, and on matrix formats for linear algebra workloads.

A. Energy-Efficient Data Movement

Data movement energy has become an increasingly pressing concern over the past decade. To address the problem, a number of techniques combining both circuit and architecture level innovations have been proposed.

1) Interconnects and Signaling: Interconnects can be broadly classified as terminated vs. unterminated; both classes are used in modern computer systems. Terminated interconnects incorporate a termination resistor on the receiver side of the interconnect, labeled R_T in Figure 1a. The termination resistor is used to match the impedance of the receiver to that of the interconnect to reduce reflections, improving signal integrity. When a **1** is to be sent by the transmitter, a conducting path is formed between V_{DD} and ground, as indicated by the dotted line. Consequently, transmitting a **1** consumes energy, while transmitting a **0** is effectively free. Although Figure 1a shows a V_{DDQ} terminated interconnect where the termination resistor is connected to V_{DD}, a terminated interconnect can also be V_{SSQ} terminated where the resistor is connected to ground. For specificity, and without loss of generality, we assume V_{DDQ} termination in the rest of this paper.

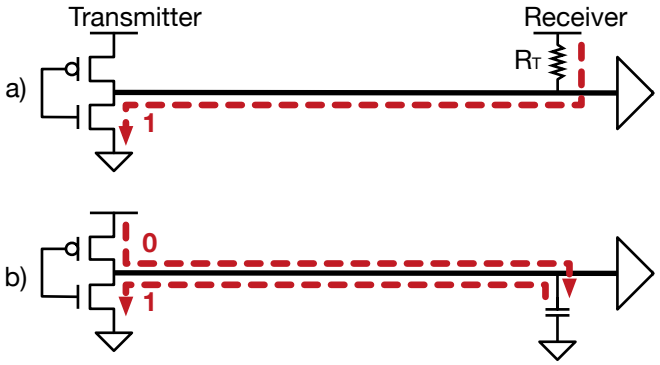


Fig. 1: a) Example of VDDQ terminated interconnect; b) example of unterminated interconnect.

Unterminated interconnects (Figure 1b) have no termination resistor; therefore, no direct path from V_{DD} to ground exists. Instead, unterminated interconnects consume energy by charging and discharging the capacitance of the interconnect and the attendant gates, consuming energy whenever the state of the bus transitions from a **1** to a **0** and back. Notably, it is possible to make the energy consumption on an unterminated interconnect proportional to the number of **1**s in the transmitted data by adopting a *transition signaling* convention [10]. In transition signaling, a **1** is signaled from the transmitter to the receiver by toggling the state of the wire that connects the two circuits, and a **0** is signaled by maintaining the state of that wire. Consequently, transmitting a **1** consumes energy, whereas a **0** is effectively free.

High-speed off-chip interconnects, including those specified by the DDR3/4, LPDDR3/4, and GDDR4/5/5X standards, allow for termination to improve signal integrity [3], [5], [6]. On-die interconnects are often unterminated to reduce complexity and layout size. Unterminated interconnects are also used in the new 3D stacked DRAM standards such as High Bandwidth Memory (HBM) [2] and Wide-I/O [4]. These standards use wide buses of 512 to 1024 bits instead of the 32-64 of DRAM, allowing them to achieve high bandwidth at much lower signaling rates than GDDR5/5X.

2) Energy Efficient Data Encoding: The asymmetric energy costs discussed above create significant potential for data representations and encodings that aim to reduce the number of **1**s. One early scheme for efficient data representation was Bus-Invert Coding (BIC) [44], standardized as Data Bus Inversion (DBI) in DDR4 [6]. DBI observes that if more than $\frac{N}{2}$ bits of an N-bit data block are **1**s, the data can be inverted—and a one-bit flag set to indicate the inversion—such that at most $\lceil \frac{N}{2} \rceil$ bits would be **1**s. DBI has been adopted by several recent DRAM standards for both terminated and unterminated interconnects. In these standards, DBI is applied at the byte

granularity such that a 64 bit DRAM data bus requires eight additional wires.

Beyond DBI, several papers have leveraged coding theory to represent data using fewer **1**s at the cost of wider transmissions. One such technique applies M-limited weight codes (LWCs) to data transmissions [45], encoding a data word such that only M bits are **1**s. With this scheme, each transmitted codeword must be significantly longer than the original data; for example, representing a byte requires 17 bits with a 3-LWC. More is Less Coding (MiLC) [43] is an application of 3-LWCs that exploits DRAM bus underutilization to opportunistically encode transmissions when the wider operands would not substantially degrade performance.

Another set of techniques attempts to exploit data similarity and value locality to reduce data movement energy. Since applications exhibit structure within both accesses and datasets, certain bit patterns are more likely than others, creating the potential to represent frequent values with fewer **1**s. One approach to energy efficient data encoding is to maintain a table of recently seen data values, and to transmit the bitwise XOR between the new value and one of the previously observed patterns. This approach requires the index of the earlier pattern to be transmitted alongside the value to enable decoding at the receiver, and has been employed by several proposals including Frequent Value encoding [52] and Bitwise Difference (BD) encoding [42]. An extension of this approach was proposed by Wang *et al.* [50]; rather than using the previously seen data values, the authors use online k-majority clustering to dynamically learn the bases for the encoding. Typically, these schemes make use of the wires used for DBI to transmit the additional metadata required for decoding.

One weakness of these previous proposals is the requirement to modify both the DRAM DIMM and the interface. An alternative approach was recently proposed by Lee *et al.* [32] where the data is encoded before being written to the DIMM, and subsequently read in this encoded state to be decoded by the memory controller. This means that no additional metadata or modifications to the DIMM are required, thereby allowing DBI to be used alongside Base+XOR to lower the cost of the unencoded base and data in blocks with low similarity. Instead of using a table of bases, the first word in a cache block is transmitted unencoded, and each subsequent word is encoded as the bitwise XOR with the previous word. Lee *et al.* [32] also propose two modifications to this simple Base+XOR approach to avoid challenges due to data size and zero values.

Prior work by Childers *et al.* [15] proposes cache line reordering (CLR), a pure hardware technique where different words in a single cache line are fetched in an order that minimizes the number of transitions on the memory bus. CDR differs from CLR in two significant ways. First, CLR requires caching an ordering vector for each cache line at the memory controller. When scanning over a 128 MB sparse matrix on a system with 128B cache lines, this table would require 1M entries. CDR avoids this overhead by modifying the order in which the application stores the data in memory so that no additional metadata is required. Second, CLR restricts the ordering to a single cache line, which limits the number of possible orders—and by extension, the potential benefits. CDR allows reordering within a cache line without performance loss, and can improve the energy savings further by potentially reordering across the entire sparse matrix data structure. Although a number of restrictions must be placed on which coefficients can be reordered, CDR generally considers reordering well beyond a single cache line.

B. Linear Algebra for Inference Tasks

Linear algebra kernels are at the heart of modern neural network inference tasks. For instance, in convolutional neural networks (CNNs), the convolution layers can be implemented as matrix-matrix multiplication operations, and fully connected layers as matrix-vector multiplication operations.

Over the past several years, there has been significant attention on the concept of *pruning* to reduce the bandwidth and energy costs of fetching the weight matrices from memory. Although weight matrices in fully connected neural network layers are typically dense—every input of a layer connects to every output—recent work has found that many of these weights can be removed without impacting accuracy [22], [24], [38], [39], [53]. Pruning selects low weight connections to remove based on a variety of heuristics and metaparameters, retraining the network around the eliminated connections. Since zero values have no impact on the results of the computation, they can be safely discarded at the cost of additional metadata. Numerous formats have been proposed to encode the metadata based on the specific behaviors of the platform.

Pruned neural networks are essential for mobile inference tasks due to the resource constraints on mobile devices. Although weight matrices of several hundred megabytes will often fit within the main memory of a mobile device, Wu *et al.* [51] note that code size, which includes the size of neural networks, is a major limitation. Narang *et al.* [38] indicate that state of the art RNNs for translation are often 250-500MBs, a significant amount of space for a mobile device with only 32 or 64GB of storage. As more applications rely on neural networks, these storage restrictions may become even more significant. Beyond storage, Han *et al.* [22] note that larger mobile applications may require a Wi-Fi connection for updates, limiting the speed with which new versions can be deployed. Wu *et al.* [51] note that for neural networks deployed as part of augmented reality applications, Facebook currently uses a Deep Compression [22] like system to reduce transmission costs. Notably, even if the pruning does not reduce inference latency on the edge device, it still adds significant value due to the reduced storage requirements.

1) Sparse Matrix Formats: On CPU and GPU-based systems, pruned neural network tasks are often implemented using sparse matrix formats originally designed for scientific computing. In these formats, the metadata typically encodes both the row and column indeces of each non-zero to enable skipping zero values. Numerous sparse matrix formats have been proposed that each exploit different matrix and hardware properties. A detailed survey of matrix formats for sparse

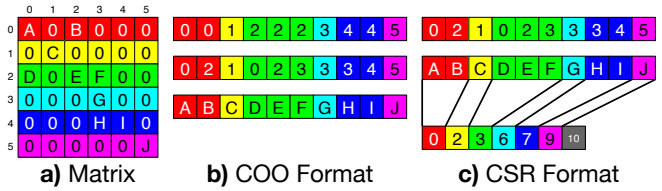


Fig. 2: Example of a sparse matrix in two different sparse matrix representations.

MVM can be found in [49], and an analysis of different formats on GPUs can be found in [12], [13]. This paper focuses on the widely used compressed sparse row (CSR) format; however, CDR can also be applied to other formats, including ELLPACK [14] and its derivatives.

Coordinate (COO) Format. Coordinate (COO) is the simplest sparse matrix format. In COO, a non-zero is represented by a three-element <row, column, value> tuple. Figure 2b shows the COO representation of the matrix in Figure 2a: the top two arrays are the row and column fields, and the bottom array is the value field such that the i^{th} element of each array gives the <row, column, value> tuple. Although COO is conceptually straightforward, the explicit storage of row and column indices increases the bandwidth requirements of the format—a hindrance in sparse MVM, which is generally bandwidth limited. Notably, although COO makes no ordering assumptions about the tuples, many COO kernels assume that the values are ordered in a row or column major order for performance reasons [12], [13].

Compressed Sparse Row (CSR) Format. Compressed Sparse Row (CSR) reduces the bandwidth requirements of COO by eliminating repetitions of the same row index when the matrix is stored in a row-major order. Instead of storing an explicit row index for each non-zero, CSR stores a row pointer array (the bottom array in Figure 2c) while maintaining the column and value fields from COO. The row pointer array stores the index of the first element in each row. Thus, the row coordinate of the i^{th} tuple is the index of the largest row pointer entry with a value less than or equal to *i*. For example, in Figure 2c, the 5th tuple <2, 2, E> is in row 2 since it falls between the second and third entries in the row pointer array.

CSR is one of the most widely used sparse matrix formats due to its ability to efficiently store unstructured sparse matrices, and ease of parallelization. On CPUs, CSR based MVM is typically parallelized by giving each thread a row, or set of rows, such that the working set matches the cache size [49]. For GPUs, Bell and Garland [12], [13] identify two different MVM kernels; the first, which they call CSR-Scalar, is similar to the aforementioned approach adopted on CPUs where each thread is responsible for one row. CSR-Scalar results in poor memory access patterns on GPUs, and an alternative, CSR-Vector, with one row per warp[1] is favored.

A column-major variant of CSR, compressed sparse column (CSC), is used in applications where the matrix is accessed in a column-major order. Since CSR and CSC share many of the same data layout traits, we discuss only CSR; however, the analyses presented in the paper are equally applicable to CSC.

[1]We adopt NVidia's terminology for thread groups (warps) in this paper.

III. Key Idea: Fetching the Matrix Coefficients in an Order that Minimizes Energy

When fetching a matrix row from memory, the order in which the coefficients are fetched directly impacts the energy of the data transmissions. Data similarity based encoding techniques reduce the overall energy consumption on a specified transmission order. However, since many of these proposals rely solely on hardware mechanisms, they miss an important optimization opportunity—the order in which the application accesses the matrix coefficients. To see the potential, consider fetching a four element vector $(0001, 1111, 1110, 1001)$ (Figure 3) over an interconnect that consumes energy when transmitting a **1**. Further assume that the data is to be encoded using a Base+XOR scheme: the first coefficient is transmitted explicitly as a base, and each subsequent coefficient is XORed with the previous coefficient prior to transmission. This encoding reduces the number of **1**s by representing the data as $(0001, 1111 \oplus 0001, 1111 \oplus 1110, 1110 \oplus 1001)$, which has eight **1**s as compared to the initial ten. The encoding can be improved by reordering the vector to $(0001, 1001, 1111, 1110)$, resulting in only five **1**s.

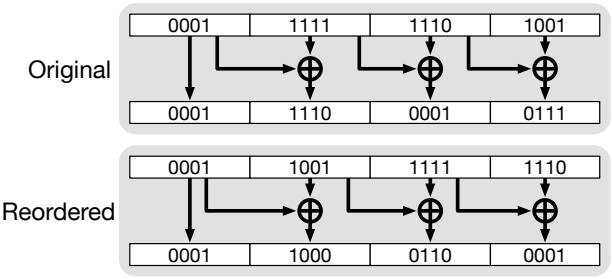


Fig. 3: Illustrative example of the benefits of data reordering.

Although this example shows significant potential, it also raises important questions. The first is whether this reordering maintains correctness. If the vector is being fetched to compute a subsequent summation, correctness is maintained since addition has the commutative property, $(a + b = b + a)$. In general, data reordering is valid whenever the fetched data is used to compute only commutative operations. Notably, many parallel applications already exploit the commutative property to partition operations among parallel threads [16], [46].

A second question is whether this reordering requires additional metadata, which may reduce the potential energy savings. For instance, if the vector in the previous example is to be used to calculate a dot product rather than a summation, the order in which the coefficients are fetched must be specified using metadata such that the appropriate vector elements can be matched prior to multiplication. Notably, prevalent sparse matrix formats used in CPU- and GPU-based systems already contain the necessary metadata since they represent the matrix as <coordinate, value> tuples (Section II-B1). CDR leverages

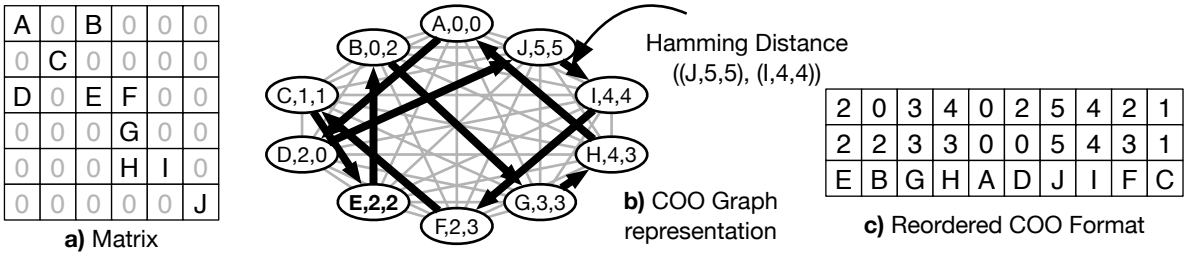


Fig. 4: Conceptual overview of Commutative Data Reordering for sparse matrix formats.

this existing metadata to support the reordering without any additional storage or bandwidth overheads.

Figure 4 shows how a sparse matrix can be reordered to reduce the cost of transmitting the matrix. In Figure 4a, there is a 6×6 matrix with ten non-zero coefficients. To model the reordering problem, the matrix is converted into a graph representation, as shown in Figure 4b. In this formulation, each non-zero coefficient is represented as a node in a fully connected graph, and each edge is annotated with a weight indicating the Hamming distance between the two nodes adjacent to it. Figure 4b also shows a graph traversal that visits every node exactly once. This traversal corresponds to a tuple ordering in COO format, as shown in Figure 4c, where node <E,2,2> is selected as the first tuple. To preserve correctness, all three values in the tuple must be reordered together; therefore, each node must contain the full tuple representing the nonzero coefficient. By comparing the COO representation shown in Figure 4c with the one shown in Figure 2b, we see that both representations have the same set of tuples, and no additional metadata is required.

Given the graph representation in Figure 4b, the problem of finding a low energy traversal is analogous to the TSP. This analogy allows us to leverage the vast-body of work on solution heuristics for the TSP to find a low cost traversal. There are a number of complications to the analogy; for instance, in Figure 4b, the traversal results in a complete cycle, whereas the transmission order in Figure 4c has a defined start and end. Furthermore, depending on the particular sparse matrix format, it may not be possible to reorder certain matrix coefficients. For example, the CSR format discussed in Section II-B1 requires each row to be stored contiguously. These complications, and how the analogy can be modified to model specific encoding schemes and sparse matrix formats, are discussed in the next section.

IV. Reordering as an Optimization Problem

The offline cost of reordering the matrix coefficients is easily amortized in neural network inference tasks, where fixed weight matrices are used repeatedly. This section explains how the problem can be modeled for optimization, and how existing optimization tools can be used to efficiently find low energy orderings.

A. Formulation as a Traveling Salesman Problem

The standard formulation of the TSP is to find a tour that visits each node exactly once and returns to the starting node. As the example in Figure 4c shows, the final matrix ordering

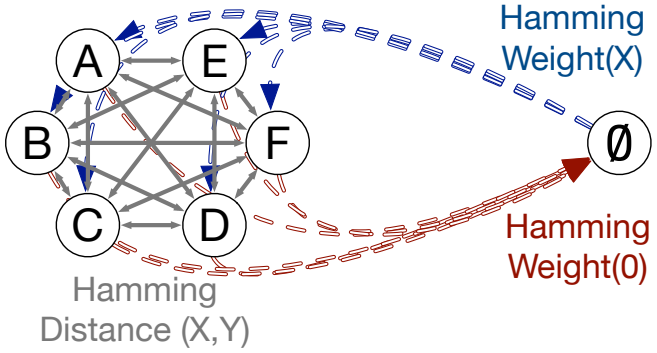


Fig. 5: Graph formulation augmented with an auxiliary node.

should not consider the cost of returning to the starting node, since the bitwise XOR between the final and first elements is not transmitted. To account for the difference, the graph is augmented with an auxiliary node ($\emptyset$ in Figure 5) with zero-cost edges incident to every other node. After a low cost tour has been found, the cycle is interpreted as a data transmission order by selecting the two neighbors of the auxiliary node as the first and last tuples to be transmitted.

To complete the formulation of CDR with an underlying Base+XOR transmission scheme (Section III) as a TSP, the auxiliary node is modified to incorporate additional information. In a Base+XOR scheme, at least one base must be transmitted unencoded (as shown in Figure 3); therefore, the optimization procedure should consider the cost of explicitly transmitting the first element. To model this, the auxiliary node is assigned asymmetric edge weights, represented by the dashed arrows in Figure 5. The distance from the auxiliary node to any other node is equal to the Hamming weight of that destination node (blue arrows), whereas the distance to the auxiliary node from any other node remains zero (red arrows). With this modification, there is an opportunity to select a low weight tuple to be the base for the transmission.

The simple problem formulation above assumes that only a single base is transmitted for the entire matrix. This is sufficient for a single-threaded application performing a linear scan over the matrix, since the coefficients are accessed sequentially. However, such a strategy would incur a significant performance penalty in a multithreaded system. Since the decoding of tuple **i** relies on knowing the value of tuple **i**-1, a naïve implementation would require every thread to read the matrix from the start to correctly decode the tuples. To avoid this prohibitive cost, a new base is transmitted at the start of each thread's portion of the matrix and optimized as a

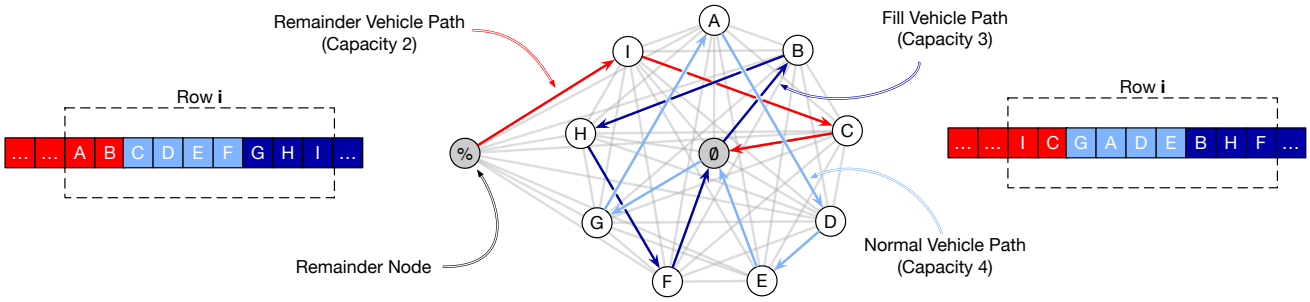


Fig. 6: CVRP formulation of matrix reordering.

separate problem with a single base (Figure 5). This naturally aligns with the restrictions imposed by the CSR format: since CSR requires tuples in the same row to be stored contiguously, the optimization problem is necessarily restricted to a single row. Notably, when reordering for CSR, each tuple contains only two fields, a value and a column ID. The row pointer array does not change since the number of tuples in each row remains unchanged after reordering.

To avoid disrupting existing memory systems, a base should be transmitted at the same granularity as data is fetched from memory, for instance a cache block [32]. Although it is possible to optimize within a single cache block, this reduces the quality of reordering by imposing restrictions beyond the format requirements. Reordering a full row with multiple bases breaks the simple analogy to the TSP; however, the problem can still be modeled using a formulation related to the TSP, the Capacitated Vehicle Routing Problem (CVRP).

B. Formulation as a Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is an extension of the TSP where instead of having a single salesman complete the tour, multiple vehicles must collectively visit every node [47]. The CVRP places a capacity restriction on the VRP such that each vehicle has a finite capacity. We model each vehicle as having capacity C, and each node as having demand 1. Therefore, a vehicle can visit at most C nodes. In order to model transmitting multiple bases within a single matrix row, C corresponds to the number of tuples in each cache block. For instance, if a system has 32B cache blocks and each operand is 2B, C for this system is 16.

Since an unencoded base is transmitted as the first element of each cache block, the auxiliary node ($\emptyset$) is the same as that shown in Figure 5. Similarly to the TSP example, each vehicle leaves the auxiliary node and completes a tour of C nodes before returning to the auxiliary node. Each row is optimized as a separate problem so as not to violate the format restrictions in CSR. This formulation accurately models multiple base transmissions within a row with two potential problems. The first problem occurs if a row does not divide evenly into the number of cache blocks. In this case, the lowest cost solution may involve multiple vehicles being underfilled, breaking the analogy of vehicles to cache blocks. To prevent this, a single *fill vehicle* is given a reduced capacity of $Row\_Length \ modulo \ C$ such that only the fill vehicle is underfilled (shown in dark blue on Figure 6).

If each row is padded to always begin on a cache block boundary, the formulation above is sufficient; however, this is not the case in many CSR implementations [12], [13]. The existence of a fill vehicle in row **i**-1 implies another reduced capacity vehicle in row **i** with capacity $C - (Row\_Length \ modulo \ C)$. This *remainder vehicle* (shown in red in Figure 6) is added to the problem formulation to enable optimization of the remaining tuples in the cache block. Unlike other vehicles where the first tuple in the route is transmitted as a base, the first tuple visited by the remainder vehicle in row **i** is encoded with respect to the final tuple in row **i**-1. Therefore, the remainder vehicle departs from a second auxiliary node (called the *remainder node* and indicated with a **%** in Figure 6) where the distance to all other nodes in row **i** is based on the Hamming distance from the final node visited by the fill vehicle in row **i**-1.

Notably, this approach does not allow the final tuple in row **i**-1 to be optimized with respect to the first tuple in row **i**. However, unless rows are very short, this is a minor limitation that substantially simplifies the problem formulation.

C. Optimizing the CVRP

To optimize the CVRP, we use the Google OR-Tools library [1]. OR-Tools is a constrained programming library that contains specialized optimizations for many versions of VRP, including the CVRP. OR-Tools contains a variety of search strategies for optimizing the CVRP, including greedy descent—which terminates when a local minimum is found—and several metaheuristics for escaping local minima. To examine the potential benefits of metaheuristics on this problem, we evaluate the transmission cost of 16 weight matrices using guided local search (GLS) [48], simulated annealing [29], and tabu search [19], [20]. OR-Tools is run on each weight matrix for 24 hours, with each row in the matrix receiving an equal share of the 24 hour period. **This offline process is performed only once during training, and is never performed on the mobile device during inference.**

The results indicate that none of the metaheuristics make a significant difference as compared to greedy descent. The largest improvement for a single layer is 1.8% over greedy descent, which comes from GLS; however, the average improvement is only 0.5% for GLS and less than 0.1% for simulated annealing and tabu search. Since applying the more sophisticated metaheuristics does not provide a significant

benefit over greedy descent, for the remainder of this paper we use greedy descent for all optimizations.

D. Optimization for Other Matrix Formats

Although the problem formulation above focuses on the basic implementation of the widely used CSR matrix format, the TSP/CVRP formulation is also applicable to a wide range of other formats. In general, any CSR/CSC derived format that does not impose ordering requirements on the values within a matrix row can be optimized with this problem formulation. Importantly, this restriction on value ordering is relatively rare in CSR/CSC derivatives. For instance, two recent proposals for optimized CSR formats, CSR-Adaptive [21] and CSR5 [34], are both compatible with CDR. Beyond these, CDR is also applicable to formats derived from ELLPACK [14], such as JDS [41] and SELLPACK [36]. However, the difference in data layout between ELLPACK and CSR requires changes to the optimization problem.

V. Conserving Performance

CDR provides the best data movement energy reduction when reordering is unrestricted, such that any coefficient of the matrix can be swapped with any other coefficient that the underlying sparse matrix format allows. Recall from Section III that in CSR, any two coefficients within the same row can be reordered without restrictions. Regrettably, such an unrestricted reordering can interfere with memory system performance, and increase the execution time. A longer execution time may in turn result in greater static energy, canceling the energy benefits of CDR.

The primary performance problem has to do with perturbing the spatial locality in the *input vector access stream* when performing a sparse MVM. If the matrix is stored in CSR format (Section II-B1), the value, column ID, and row pointer arrays are accessed sequentially regardless of the order in which the coefficients are stored. Similarly, the output vector is written one element per row, which is unaffected by the reordering within a row. However, the order in which the input vector coefficients are accessed is determined by the order in which the column IDs are encountered while scanning the column ID array. Perturbing that order may not only increase the cache miss rate, but may also hurt the coalescing process on a GPU, increasing the hit time.

A. Strided Reordering

In strided reordering, consecutive elements of the original sparse matrix are grouped into *strides* within which the reordering is performed. Each stride begins at the start of the row and stops either at the end of the row or after stride length elements. For example, Figure 7 shows how three rows, indicated by the colored regions, are grouped into strides of 4 elements. Since row_1 has five elements, the first four elements are placed in one stride, and the final element **H** is placed in another stride. This restriction makes it possible to retain at least some of the spatial locality within the original vector access stream. In general, the greater the stride length, the greater the possibility of a reordered vector access stream with adverse characteristics.

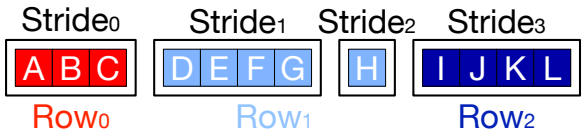


Fig. 7: Example of strided reordering with a stride size of four.

1) Choosing a Stride Size: The stride size is a tunable parameter that allows the system designer to tradeoff the quality of the reordering against the performance loss from perturbations to the input vector access stream. This tradeoff can be quantified through characterization of the system at a variety of different stride sizes. Additionally, for the purposes of characterization, it is not necessary to perform the full optimization out to a local minimum. Instead, the optimization library can return the first solution found regardless of ordering quality, which provides sufficient information for the purposes of selecting a stride size. To reduce the characterization overhead, we observe that stride sizes that are an integer multiple of the number of threads per warp tend to have lower performance penalties than similar stride sizes that are not. Therefore, the characterization need only be performed on these integer multiple stride sizes.

2) Integrating Stride Size into the Optimization Problem: Once a stride size has been selected, it is integrated into the CVRP formulation discussed in Section IV-B. The stride size is analogous to splitting each row into multiple sub-rows of stride length, with each sub-row as a discrete optimization problem. This increases the potential loss from not optimizing the final tuple of sub-row N-1 with respect to the first tuple of row N; however, for most stride lengths, the effects are small. Additionally, as the sub-rows get smaller, the potential improvements from this optimization decrease due to the reduced number of potential tuple pairings.

VI. Hardware Support

When CDR is applied to a GPU-based system, no hardware overhead is required beyond an encoder and decoder for the underlying data similarity based encoding scheme.

Existing encoding techniques proposed by Wang *et al.* [50] and Lee *et al.* [32] target the interconnects between main memory and the last-level cache (LLC). The proposed CDR technique is orthogonal to these encoding schemes, as it targets the scheduling of data movement rather than encoding the data itself. For simplicity, this paper adopts a Base+XOR encoding scheme [32]; however, CDR is also compatible with other data similarity based encoding schemes.

When applying CDR to sparse MVM where the matrix is stored in CSR format, not all of the data can leverage CDR. CDR enables reordering of the value and column ID arrays; however, the row pointer array in the matrix is not reordered. Similarly, the input and output vectors cannot be reordered with CDR since there is no indirection in the vectors to hold the metadata. In practice, this is not a significant limitation,

since the majority of the data is contained in the two arrays that are reordered.

The aforementioned observation prevents us from significantly modifying the encoding scheme based on the capabilities of CDR since some portion of the transmitted data does not leverage CDR. For instance, the Base+XOR encoding proposed by Lee *et al.* [32] remaps all-zero words to a low-weight value to prevent increasing the number of **1**s when transmitting zero values interleaved with non-zeros. With CDR, this pathological case of Base+XOR encoding is rare; however, we cannot remove this special zero handling because it may be relevant to transmitted data not amenable to CDR.

A. Integration with Memory System Optimizations

Modern memory systems often implement optimizations that can reduce the energy consumption of the interface, and improve the signal integrity. CDR can be integrated with these memory system optimizations.

1) Data Bus Inversion: As discussed in Section II-A2, data bus inversion (DBI) is a technique to reduce the power consumption of the interconnect by selectively inverting each byte if more than half of the bits in the byte are **1**s. In addition to reducing power consumption, DBI also reduces power noise, increasing signal integrity by capping the number of transmitted **1**s and suppressing simultaneous switching noise. Due to these benefits, DBI is part of several recent DRAM standards, including DDR4 [6], LPDDR4 [3], and GDDR5 [5].

CDR is fully compatible with DBI since CDR requires no metadata or modification to the underlying encoding scheme. Beyond compatibility, DBI can be incorporated into the optimization problem to further reduce the cost of transmission. To allow the optimization problem to consider DBI, the Hamming weights and Hamming distances discussed in Section IV must be modified to include DBI at a byte granularity. All evaluations in this paper include DBI, and DBI-incorporating distances are used for all optimizations.

2) Data Scrambling: Data scrambling is an alternative to DBI that reduces simultaneous switching noise and broadens the frequency spectrum of transmissions by XORing the data with a pseudo-random value [37]. Similar to other energy-efficient data encoding proposals [32], [40], [42], [43], [50], CDR is not directy compatible with data scrambling, and instead assumes the DBI mechanism adopted by DDR4 [6], LPDDR4 [3], and GDDR5 [5]. As Pekhimenko *et al.* [40] explain, data scrambling "contradicts what several designs aim to achieve by using DBI for GDDR5 [5] and DDR4 [6], since data scrambling causes the bits to become more random." Nevertheless, CDR and data scrambling can coexist in a system in which the appropriate technique is selectively enabled based on application needs.

TABLE I: GPU Architecture Configuration

System	256-Core Pascal @1300MHz, 8 SMs, 32 threads/warp
SM	32 warps/SM, 32K registers, 48KB Shared Memory, 24KB L1 Data Cache.
L2 Cache	512KB
Memory	8GB LPDDR4 @ 1866MHz, Max Bandwidth 59.7GB/s 4 Memory controller with FR-FCFS scheduling, 16 banks per controller.

VII. Experimental Setup

Evaluating CDR requires modeling the performance and energy characteristics of pruned neural networks on mobile-class GPU platforms.

A. Architecture

To evaluate CDR on GPU-based systems, we use the cycle accurate simulator GPGPU-Sim [11] modified to simulate a LPDDR4 memory standard, with the system parameters listed in Table I. The configuration parameters are representative of a mobile-class Nvidia Tegra X2 GPU [7]. The DRAM interface energy is modeled using the Micron LPDDR4 power calculator [35].

TABLE II: Evaluated Networks

Name	Base Network	Pruning Approach
an_dc	AlexNet [31]	Deep Compression [22]
an_skim	AlexNet	SkimCaffe [39]
an_ip	AlexNet	Iterative Pruning [24]
mn_agp	MobileNet [26]	Automated Gradual Pruning [53]
rn_agp	ResNet [25]	Automated Gradual Pruning
vd_agp	VD-LSTM+REAL [27]	Automated Gradual Pruning
vd_bd	VD-LSTM+REAL	Baidu RNN Pruning [38]

B. Applications

To evaluate the energy savings on inference workloads, we use the seven neural networks specified in Table II. Three of the networks are implementations of the widely used AlexNet [31] structure, with three different pruning methods applied to generate different sparsity and data patterns. All three AlexNet implementations are pre-trained. The next two neural networks are two commonly used networks for image classification, ResNet [25] and MobileNet [26]. Both use the pre-trained weights provided by the Distiller framework [54]. Additionally, we evaluate two recurrent neural networks (RNNs) using the Distiller framework for pruning.

For each network, the weight matrices are extracted as 32-bit floating point values and converted to both 16-bit floating point and eight-bit integer representations. The matrices are re-ordered using Google OR-Tools as described in Section IV-C, where new rows are allowed to begin within a previous stride.

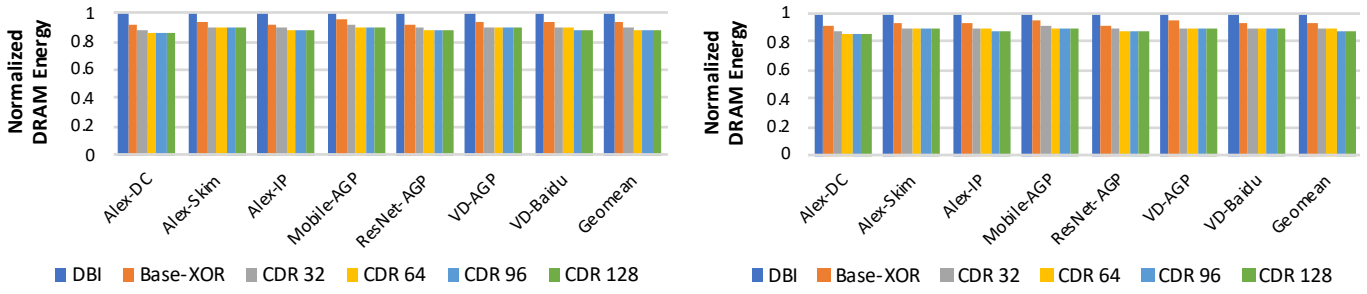
The sparse-MVM kernel is based on the CSR-vector kernel in the Scalable Heterogeneous Computing (SHOC) benchmark suite [18]; however, for generalizability to other architectures, we use the L1 data cache rather than the texture cache for vector accesses. For convolutional layers, we use the direct sparse convolution approach proposed by Park *et al.* [39], where the convolution is computed as a series of MVMs.

VIII. Evaluation

The area, energy, and performance characteristics of CDR are evaluated in this section.

A. Energy

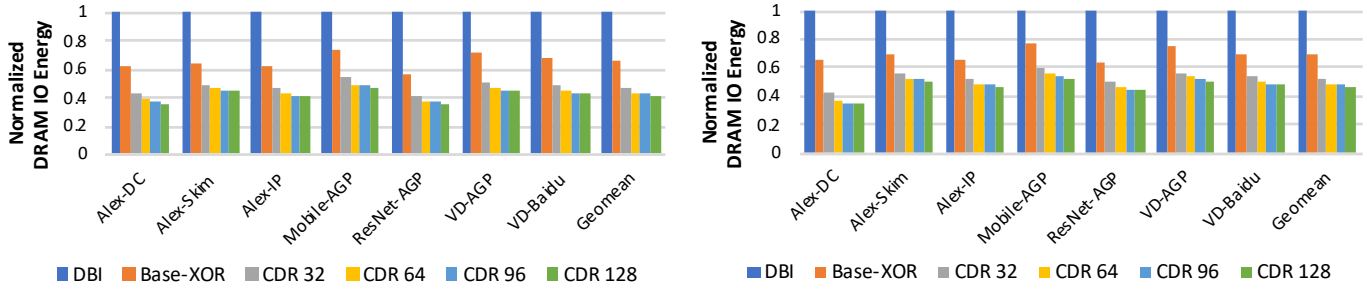
The energy consumption of the LPDDR4 system using half-precision floating point and eight-bit fixed point is shown in Figure 8. Over for the set of evaluated stride sizes, when CDR is combined with the previously proposed Base+XOR



(a) Fixed Point.

(b) Floating Point.

Fig. 8: Normalized DRAM energy for fixed and floating point.



(a) Fixed Point.

(b) Floating Point.

Fig. 9: Normalized DRAM I/O energy for fixed and floating point.

encoding, the DRAM energy consumption is reduced by up to 13% over an interconnect using DBI alone.

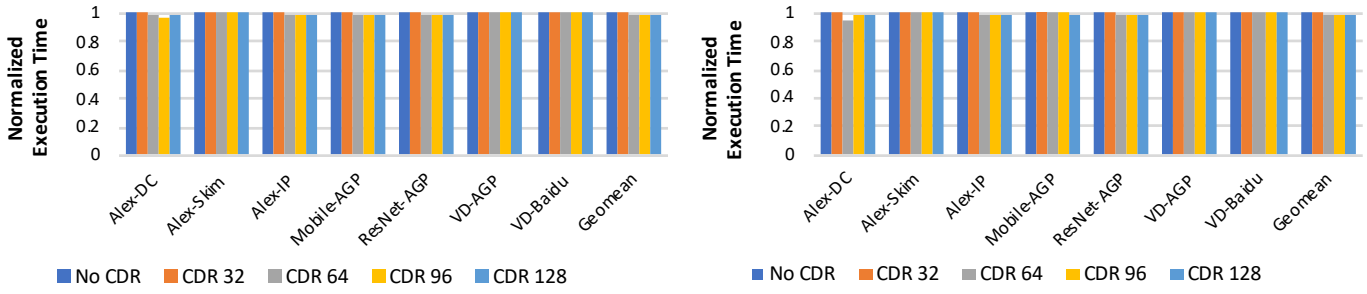
If we confine the energy evaluation to stride sizes with no reduction in performance relative to the unreordered matrix, we find an average improvement of 12.7% in DRAM energy compared to a system using only DBI, and 5.3% compared to Base+XOR alone. The reduction in DRAM energy is a direct result of the reduction in DRAM IO energy shown in Figure 9, where an up to 53.1% reduction in DRAM IO energy is achievable—a 22.2% reduction over Base+XOR, again considering just stride sizes with no performance penalty. The 5.7% reduction in average DRAM energy—and up to an 8.8% reduction—over existing schemes is comparable to the improvements reported by prior work on energy efficient data encoding schemes [32], [42], [43], [50]. Base+XOR encoding [32] for instance reports a 2.9% reduction in DRAM energy over BD coding [42]. Similarly, MiLC [43] found an 8% reduction in DRAM energy over DBI alone, and online K-majority clustering [50] found a 4% reduction over frequent value coding, and a 9% reduction over DBI.

One network that stands out is Deep Compression Alexnet. The Deep Compression technique not only prunes the matrix but also quantizes the coefficients into a smaller number of floating point values. This improves the encoding with and without CDR when using Base+XOR encoding. The repetition of values in these weight matrices make them especially amenable to reordering with longer stride sizes. The larger improvement on this neural network also shows the synergy between CDR and neural network compression techniques that optimize coefficient values as well as sparsity patterns.

The energy results above are significant given the high cost of data movement in general and DRAM accesses in particular. The high cost of data movement has been noted in CPUs [42], [43], [50] and GPUs [28], [40], with DRAM energy consumption ranging from 20% to 40% of system power. For mobile inference workloads, the importance of DRAM IO energy is likely to become more pronounced. The recently announced LPDDR5 standard [9] is designed for up to 6400MT/s, doubling the 3200MT/s of LPDDR4. As Lee *et al.* [32] indicate, significant increases in memory bandwidth are likely to increase the DRAM IO energy. Furthermore, given the interest in neural networks from both academia and industry, we are likely to see widespread adoption of dedicated datapaths in conventional SOCs and specialized inference accelerators, some of which are in modern mobile SOCs today [51]. These dedicated datapaths and accelerators reduce compute energy, further amplifying the importance of energy-efficient data movement.

B. Performance

Figure 10 shows the performance impact of CDR. Using striding, we can find a stride size for each neural network where there is no performance loss from inference. In general, over this set of stride sizes, the maximum performance loss is 2.1% and all other performance penalties are under 1.1%. Notably, convolutional layers appear less sensitive to striding than fully connected layers due to the smaller matrix size and greater data reuse. In fact, convolutional layers are more likely to benefit from the prefetching effects of matrix reordering discussed in Section V-A. The overall effect is small, typically less than 5%; however, since convolutional layers constitute the bulk of the runtime in CNNs, the performance improvements on these layers effectively balance the potential performance losses on fully connected layers.



(a) Fixed Point.

(b) Floating Point.

Fig. 10: Normalized execution time for fixed and floating point.

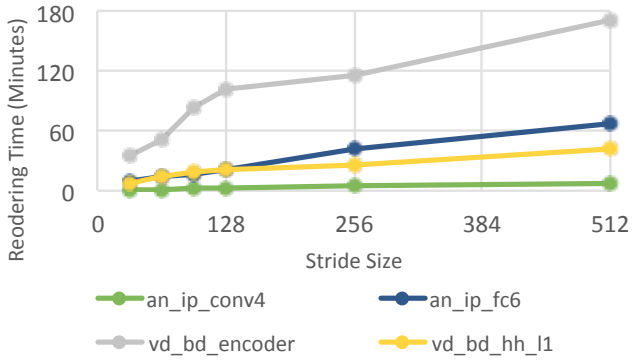


Fig. 11: Offline optimization time for greedy descent as a function of stride size.

Overall, these results show that when appropriate striding is used, CDR can be implemented with minimal performance impact, and in some cases a small performance increase. Additionally, these results are consistent between both fixed and floating point reordering patterns, suggesting broad applicability to inference workloads.

C. Area

The evaluated system requires small modifications to a GPU. Since the Base+XOR codecs proposed by Lee *et al.* [32] were evaluated on the same 16nm process as the modeled GPU, we use the reported codec area of $2232\mu m^2$ for a direct comparison. The area overhead of adding a codec to all four memory controllers is less than 0.01% of the total SOC area. Conservatively estimating that a codec for a 128B cache line would require $8\times$ the area of a 32B codec, the total area required to enable CDR is 0.02%.

D. Offline Optimization Time

Figure 11 shows the offline optimization time for greedy descent as a function of stride size, over four weight matrices from the test set. These matrices cover one of each type of weight matrix to show the variation in optimization time. The optimization is performed on a system with a 2.4GHz Intel Xeon E5-2630 processor with 32GB of RAM.

Even in the longest case, vd_bd_encoder, the total optimization time for a stride size of 512 elements is less than 3 hours. Given that state of the art RNNs for speech recognition can take days or weeks to train [38], even this worst case reordering time is acceptable. However, if the reordering needs to be accelerated, there are two possibilities. First, rather than returning a local minimum for each row, the optimization process could terminate after a sufficiently good order is found. Second, this weight matrix has 33278 rows; therefore, the optimization time per row is approximately 0.31 seconds, suggesting potential for parallelizing this optimization.

E. Sensitivity to Reordering Parameters

The impacts of different CDR parameters and data formats are evaluated in this section. For readability, the results of a representative set of the weight matrices are shown rather than the full neural networks.

1) Sensitivity to Stride Size: As discussed in Section V-A, striding creates a tradeoff between the quality of reordering and performance. Figure 12 shows how longer stride sizes impact the reordering quality. The greatest improvement with larger stride sizes are the weight matrices compressed using the Deep Compression technique [22]. As mentioned above, Deep Compression quantizes the weights to reduce the number of unique coefficients in a weight matrix, thereby reducing the bit entropy. This allows the matrix to take advantage of longer strides by packing runs of identical coefficients together.

In general, the average improvement from the smallest evaluated stride to unrestricted reordering is 13.1% over the the unreordered matrices, and 11.4% excluding the Deep Compression matrices mentioned above. Interestingly, neither sparsity nor matrix size appear to have a predictable influence on sensitivity to stride size. Comparing between similar weight matrices, for instance, the conv4 matrices from the iterative pruning and AGP Alexnet networks exhibit both the 2nd largest and the 2nd smallest improvement with increasing stride sizes. This suggests that the underlying sparsity pattern plays an important role in the reordering quality and may have some additional potential for further reordering aware pruning.

Figure 14 shows the impact of stride sizes on the number of L1 cache accesses and misses when performing MVM on the an_dc_fc7 layer. As discussed in Section V, striding makes the accesses to the input vector more scattered, which hampers coalescing on a GPU and results in a greater number of accesses to the L1 arrays. In turn, this increases the hit time for other memory requests. The miss rate increases gradually with stride size; since multiple warps often need input vector

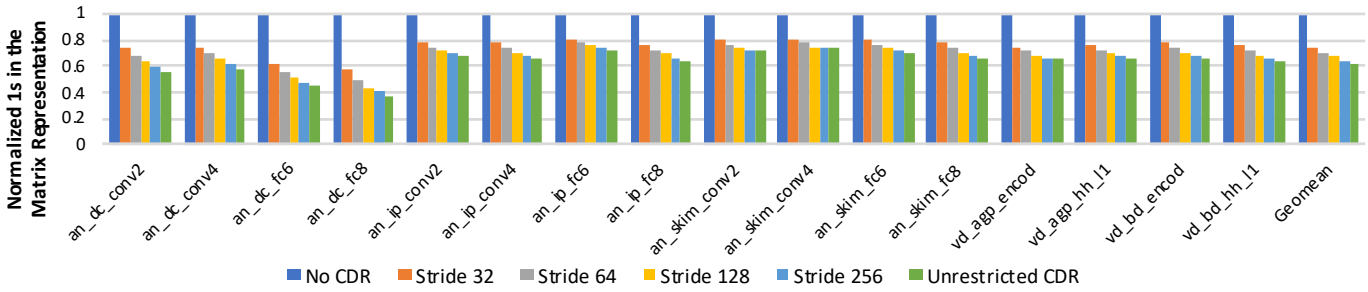


Fig. 12: Sensitivity to Stride Size.

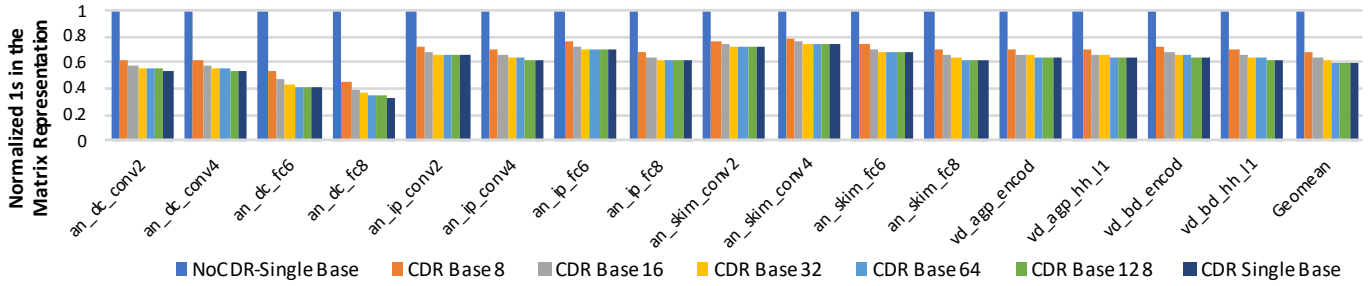


Fig. 13: Sensitivity to Base Retransmission.

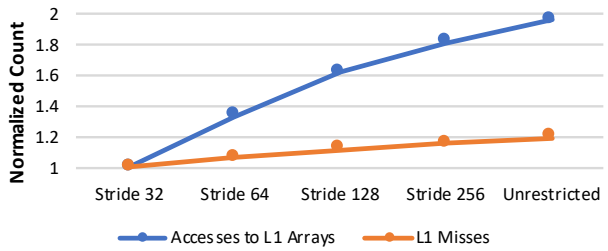


Fig. 14: L1 cache behavior for an_dc_fc7 with multiple stride sizes.

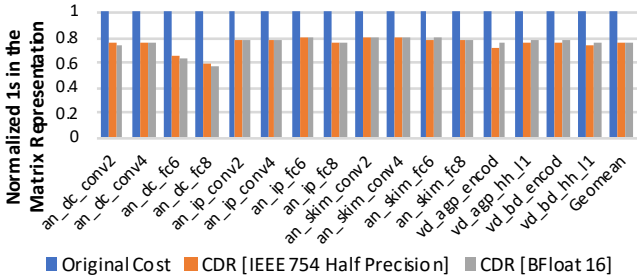


Fig. 15: Sensitivity to Floating Point Format.

elements in the same cache block, the scattered accesses can effectively prefetch input vector elements. At smaller stride sizes, this prefetching effect can counteract the increase in hit time and mitigate the performance loss, or even slightly improve performance as discussed in Section VIII-B.

2) Sensitivity to Base Retransmission Frequency: Prior work on Base+XOR encoding did not investigate the effects of reducing the frequency with which base values are transmitted unencoded. As noted in Section IV, the base retransmission frequency is largely determined by the underlying hardware. Figure 13 shows how the cost of matrix transmission scales with increasing base retransmission frequency. This shows that the benefits of CDR are substantial even with frequent base retransmissions. Compared to a system where the base is transmitted only once for the entire matrix, CDR still achieves an 32% improvement in the number of transmitted **1**s over Base+XOR encoding alone. This is due to the problem formulation shown in Figure 5, where reordering also attempts to place a tuple with low Hamming weight in the base position.

3) Sensitivity to Floating Point Format: Recent work on data formats for neural networks has found that the half-precision format specified by IEEE-754 floating point may not be the best option for these workloads. One format that has recently seen significant attention from industry is bfloat16, a 16-bit floating point format with eight exponent bits instead of five with IEEE-754 half precision [8]. Figure 15 shows that CDR has similar impact on both half precision and bfloat16 formats. In fact, CDR may be more beneficial on systems using bfloat16, since bfloat16 values have 9.4% more **1**s when encoded using Base+XOR.

IX. Conclusion

Prior work on energy-efficient encoding schemes has achieved significant reductions in energy by exploiting the similarity inherent to the data transmitted over the interconnect. To continue improving data movement energy using encoding techniques, we need to go beyond considering data as a mere stream of bits, and leverage application level semantics. This paper presents the first proposal to exploit these application level characteristics. The proposed commutative data reordering approach reorders sparse neural network weight coefficients offline to optimize the order in which data are fetched from memory. Commutative data reordering reduces the DRAM IO energy by 22.2% over the recently proposed Base+XOR encoding. These savings correspond to total DRAM energy savings of 5.3%, and are attained at no performce cost. In this era of increasing specialization, the potential for encoding schemes tailored to specific applications is especially promising.

REFERENCES

[1] Google's OR-Tools 7.0. Google. [Online]. Available: https://developers.google.com/optimization/

[2] "High bandwidth memory (HBM) DRAM," JEDEC Solid State Technology Association, Tech. Rep. JESD235, Oct 2013.

[3] "Low power double data rate 4 (LPDDR4)," JEDEC Solid State Technology Association, Tech. Rep. JESD209-4, Aug 2014.

[4] "Wide I/O 2 (WideIO2)," JEDEC Solid State Technology Association, Tech. Rep. JESD229-2, Aug 2014.

[5] "Graphics double data rate (GDDR5) SGRAM standard," JEDEC Solid State Technology Association, Tech. Rep. JESD212C, Feb 2016.

[6] "DDR4 SDRAM standard," JEDEC Solid State Technology Association, Tech. Rep. JESD79-4B, June 2017.

[7] "Tegra X2 NVIDIA parker series SoC," Nvidia, Tech. Rep. DP-07281-001_v1.0p, June 2017.

[8] "Bfloat16 - hardware numerics definition," Intel Corporation, Tech. Rep. 338302-001US, Nov 2018.

[9] "Low power double data rate 5 (LPDDR5)," JEDEC Solid State Technology Association, Tech. Rep. JESD209-5, Feb 2019.

[10] M. Anders, N. Rai, R. K. Krishnamurthy, and S. Borkar, "A transition-encoded dynamic bus technique for high-performance interconnects," *IEEE Journal of Solid-State Circuits"*, vol. 38, no. 5, pp. 709–714, May 2003.

[11] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, April 2009.

[12] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corporation, Tech. Rep. NVR-2008-004, Dec 2008.

[13] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Intl. Conf. on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2009.

[14] T. C. Oppe and D. Kincaid, "The performance of ITPACK on vector computers for solving large sparse linear systems arising in sample oil reservoir problems," *Communications in Applied Numerical Methods*, vol. 3, pp. 23 – 29, 01 1987.

[15] B. R. Childers and T. Nakra, "Reordering memory bus transactions for reduced power consumption," in *Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, June 2000.

[16] A. T. Clements, M. F. Kaashoek, N. Zeldovich, R. T. Morris, and E. Kohler, "The scalable commutativity rule: Designing scalable software for multicore processors," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 4, p. 10, 2015.

[17] B. Dally. (2015, Jan) Challenges for future computing systems. [Online]. Available: "https://www.cs.colostate.edu/~cs575dl/Sp2015/Lectures/Dally2015.pdf"

[18] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, March 2010.

[19] F. Glover, "Tabu search—part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[20] F. Glover, "Tabu search—part II," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[21] J. L. Greathouse and M. Daga, "Efficient sparse matrix-vector multiplication on gpus using the csr storage format," in *Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2014, pp. 769–780.

[22] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Intl. Conf. on Learning Representations (ICLR)*, May 2016.

[23] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Intl. Symp. on Computer Architecture (ISCA)*, June 2016.

[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Intl. Conf. on Neural Information Processing Systems (NIPS)*, Dec 2015, pp. 1135–1143.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[27] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," in *Intl. Conf. on Learning Representations (ICLR)*, April 2017.

[28] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sept 2011.

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[30] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. M. Bhandarkar, W. Matusik, and A. Torralba, "Eye tracking for everyone," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Intl. Conf. on Neural Information Processing Systems (NIPS)*, Dec 2012.

[32] D. Lee, M. O'Connor, and N. Chatterjee, "Reducing data transfer energy by exploiting similarity within a data transaction," in *Intl. Symp. on High Performance Computer Architecture (HPCA)*, Feb 2018.

[33] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[34] W. Liu and B. Vinter, "Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Intl. Conf. on Supercomputing (ISC)*, June 2015.

[35] Micron Technology, "LPDDR4 power calculator."

[36] A. Monakov, A. Lokhmotov, and A. Avetisyan, "Automatically tuning sparse matrix-vector multiplication for gpu architectures," in *Intl. Conf. on High Performance Embedded Architectures and Compilers (HiPEAC)*, 2010.

[37] P. Mosalikanti, C. Mozak, and N. Kurd, "High performance DDR architecture in Intel® Core™ processors using 32nm CMOS high-K metal-gate process," in *Intl. Symp. on VLSI Design, Automation and Test (VLSI-DAT)*, April 2011.

[38] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," in *Intl. Conf. on Learning Representations (ICLR)*, April 2017.

[39] J. Park, S. Li, W. Wen, P. Tang, H. Li, Y. Chen, and P. Dubey, "Faster cnns with direct sparse convolutions and guided pruning," in *Intl. Conf. on Learning Representations (ICLR)*, April 2017.

[40] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "A case for toggle-aware compression for GPU systems," in *Intl. Symp. on High Performance Computer Architecture (HPCA)*, March 2016.

[41] Y. Saad, "Krylov subspace methods on supercomputers," Research Institute for Advanced Computer Science, NASA Ames Research Center, Tech. Rep. NASA-CR-185419, December 1988.

[42] H. Seol, W. Shin, J. Jang, J. Choi, J. Suh, and L.-S. Kim, "Energy efficient data encoding in DRAM channels exploiting data value similarity," in *Intl. Symp. on Computer Architecture (ISCA)*, June 2016.

[43] Y. Song and E. Ipek, "More is Less: Improving the energy efficiency of data movement via opportunistic use of sparse codes," in *Intl. Symp. on Microarchitecture (MICRO)*, Dec 2015.

[44] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 1, pp. 49–58, 1995.

[45] M. R. Stan and Y. Zhang, "Perfect 3-limited-weight code for low power I/O," in *Intl. Workshop on Power and Timing Modeling, Optimization and Simulation*, Sept 2004.

[46] J. G. Torborg, "A parallel processor architecture for graphics arithmetic operations," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, 1987, pp. 197–204.

[47] P. Toth and D. Vigo, Eds., *The Vehicle Routing Problem*. Philadelphia, Pennsylvania: SIAM, 2002.

[48] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469 – 499, 1999.

[49] R. Vuduc, "Automatic performance tuning of sparse matrix kernels," Ph.D. dissertation, 2003.

[50] S. Wang and E. Ipek, "Reducing data movement energy via online data clustering and encoding," in *Intl. Symp. on Microarchitecture (MICRO)*, Oct 2016.

[51] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, W. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at Facebook: Understanding inference at the edge," in *Intl. Symp. on High Performance Computer Architecture (HPCA)*, Feb 2019.

[52] J. Yang, R. Gupta, and C. Zhang, "Frequent value encoding for low power data buses," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 9, no. 3, pp. 354–384, July 2004.

[53] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[54] N. Zmora, G. Jacob, and G. Novik, "Neural network distiller," Jun. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1297430