United States Patent Application Publication 20250265038
Kind Code A1
Publication Date August 21, 2025
Inventor(s) Abali; Bulent et al.

DECODER HAVING MULTIPLE STAGES OF ADDERS TO DECODE DELTA ENCODED DATA

Abstract

Provided are a decoder unit, a decompressor unit, and method for a decoder of multiple stages of adders to decode delta encoded data. A decoder unit implemented in a cache memory having a cache memory cell array comprises a first stage of adder circuits to add, in parallel, pairs of encoded items transformed using a delta encoding, wherein the encoded items include a plurality of deltas of neighbors of sequential source items. The decoder unit further comprises at least one successive stage of adder circuits to add, in parallel in each stage, pairs of outputs from a previous stage of adder circuits, wherein each successive stage has fewer adder circuits than the previous stage, and wherein output from a last of the at least one successive stage comprises the sequential source items.

Inventors: **Abali; Bulent (Tenafly, NJ), Buyuktosunoglu; Alper (White Plains, NY)**

Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** (ARMONK, NY)

Family ID: **1000007730526**

Appl. No.: **18/442669**

Filed: **February 15, 2024**

Publication Classification

Int. Cl.: **G06F7/501** (20060101); **G06F7/498** (20060101); **H03M7/30** (20060101)

U.S. Cl.:

CPC **G06F7/501** (20130101); **G06F7/4985** (20130101); **H03M7/6011** (20130101);

Background/Summary

BACKGROUND OF THE INVENTION
[0001] 1. Field of the Invention
[0002] The present invention relates to a decoder unit, decompressor unit, and method for a decoder of multiple stages of adders to decode delta encoded data.
[0003] 2. Description of the Related Art
[0004] Data may be encoded using delta encoding which subtracts successive items in a chunk of data, such as bytes in a word of data. For instance, for source data comprising 8 bytes A**0**. . . . A**7**, the delta encoding subtracts consecutive items as: A**0-0**, A**1**-A**0**, A**2**-A**1**, A**3**-A**2**, A**4**-A**3**, A**5**-A**4**, A**6**-A**5**, A**7**-A**6** to produce encoded output B**0**, B**1**. . . . B**7**. To recover the source A values from the delta encoded items B, data is decoded sequential in the following operations, A**0**=B**0**, A**1**=B**1**+A**0**, A**2**=B**2**+A**1**, A**3**=B**3**+A**2**. . . . A**7**=B**7**+A**6**. This is the case because Bi+A(i-**1**) = (Ai-A (i-**1**)) +A (i-**1**) =Ai. Thus, to decode n delta encoded items, there are n sequential decoding operations on each successive encoded data item Bi.
SUMMARY
[0005] Provided are a decoder unit, a decompressor unit, and method for a decoder of multiple stages of adders to decode delta encoded data. A decoder unit implemented in a cache memory having a cache memory cell array comprises a first stage of adder circuits to add, in parallel, pairs of encoded items transformed using a delta encoding, wherein the encoded items include a plurality of deltas of neighbors of sequential source items. The decoder unit further comprises at least one successive stage of adder circuits to add, in parallel in each stage, pairs of outputs from a previous stage of adder circuits, wherein each successive stage has fewer adder circuits than the previous stage, and wherein output from a last of the at least one successive stage comprises the sequential source items.

Description

BRIEF DESCRIPTION OF THE DRAWINGS
[0006] FIG. **1** illustrates an embodiment of a processor chip including multi-level caches. FIG. **2** illustrates an embodiment of a cache memory in the multi-level caches.
[0007] FIG. **3** illustrates an embodiment of a compressor unit implemented in the cache memory.
[0008] FIG. **4** illustrates an embodiment of a compressed chunk package.
[0009] FIG. **5** illustrates an embodiment of a header generated for a compressed chunk.
[0010] FIG. **6** illustrates an embodiment of operations performed by the compressor unit to compress data for a cache line to store in the cache.
[0011] FIG. **7** illustrates an embodiment of an ensemble of zero-value compressors to use for the zero-value compressor in the compressor unit of FIG. **3**.
[0012] FIG. **8** illustrates an embodiment of operations to use the ensemble of zero-value compressors of FIG. **7** to compress data.
[0013] FIG. **9** illustrates a diagram of the result of compressing data with zero-value compressors having different item sizes of the data units removed for having zero values.
[0014] FIG. **10** illustrates an embodiment of a decompressor unit.
[0015] FIG. **11** illustrates an embodiment of a multi-stage delta decoder.
[0016] FIG. **12** illustrates an embodiment of a parallel delta decoding operation using the multi-stage delta decoder of FIG. **11** for encoded 8 items, such as 8 bytes of a word.
[0017] FIG. **13** illustrates an embodiment of operations performed in a multi-stage delta decoder to decode n items in a word, such as n bytes.
DETAILED DESCRIPTION
[0018] Described embodiments improve the latency of delta decoding technology to decode delta encoded items. Prior art delta decoding techniques require n sequential decoding operations of an n

item chunk to decode. Described embodiments provide a delta decoder unit having multiple stages of adders working in parallel. While total number of adders are increased, there are fewer stages of adders than the number of operations to sequentially decode the delta encoded items. For instance, the prior art requires n sequential decoding operations on each of the n items. Described embodiments provide a multi-stage decoder of adders to decode the n encoded items in log.sub.2(n) stages of parallel operations, where output of one stage is input to a next stage. Since log.sub.2(n) number of operations is substantially less than n sequential operations, described embodiments substantially reduce the latency of delta decoding.
[0019] FIG. **1** illustrates an embodiment of a processor chip **100**, in which embodiments may be implemented, including a plurality of processing cores **102**.sub.1, **102**.sub.i . . . **102**.sub.n. Each core **102**.sub.1, **102**.sub.i . . . **102**.sub.n has control logic **104**.sub.1, **104**.sub.i . . . **104**.sub.n, including core processing functions, such as Arithmetic Logic Units (ALUs) and a memory management unit, and an on-chip L1 cache **106**.sub.1, **106**.sub.i . . . **106**.sub.n and on-chip L2 cache **1081**, **108**.sub.i . . . **108**.sub.n that are private to the processing cores **102**.sub.1, **102**.sub.i . . . **102**.sub.n. The processor chip **100** further includes a last level cache (LLC) **200**, also known as an L3 cache, providing a larger storage space to cache data for the L1 and L2 caches in the different cores **102**.sub.i. There may be further levels of caches, such as an intermediate or L4 to cache data for the L3 cache **200**. The L3 cache **200** may comprise Dynamic Random Access Memory (DRAM) devices. The processing cores **102**.sub.1, **102**.sub.i . . . **102**.sub.n may write-back modified cache lines from the L2 cache **108**.sub.1, **108**.sub.i . . . **108**.sub.n to the shared last level cache (LLC) **200**, shared among the cores **102**.sub.1, **102**.sub.i . . . **102**.sub.n, to make room for a cache line evicted from the L1 cache **106**.sub.1, **106**.sub.i . . . **106**.sub.n. When modified data needs to be removed from the L2 **108**.sub.1, **108**.sub.i . . . **108**.sub.n to make room for modified data from the L1 cache **106**.sub.1, **106**.sub.i . . . **106**.sub.n. a write-back operation is performed to write the data to the last level cache **200**.
[0020] With respect to FIG. **2**, the L3 cache **200**, also referred to as a cache memory, receives cache lines from the L2 cache **106**.sub.1, **106**.sub.i . . . **106**.sub.n to write to a cache memory cell array **202** of cache lines, such as a 256 byte cache line. The cache memory **200** includes a cache memory controller **204** to manage reading and writing data to the cache memory cell array **202**, and includes a compressor unit **300** having circuitry to compress received lines of cache to allow the cache **200** to maximize the data that can be stored. A decompressor unit **1000**, described with respect to FIG. **10** below, decompresses the compressed data written to the cache memory cell array **202**. The cache memory **200** may be implemented in other caching systems than a processor chip **100** of processing cores.
[0021] The cache memory **200** may comprise a high-speed data storage layer which stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the primary storage location of the data. The cache memory **200** may comprise a volatile or non-volatile memory device, such as a Static Random Access Memory (SRAM), Dynamic Random Access Memory (DRAM), eDRAM (embedded DRAM). Other embodiments may utilize phase change memory (PCM), Magnetoresistive random-access memory (MRAM), Spin Transfer Torque (STT)-MRAM, a ferroelectric random-access memory (Efram), nanowire-based non-volatile memory, and Direct In-Line Memory Modules (DIMMs), NAND storage, e.g., flash memory, Solid State Drive (SSD) storage, non-volatile RAM, etc.
[0022] The cache memory controller **204** may be implemented in circuitry in a semiconductor device, such as a Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA) in the cache memory controller **204**.
[0023] FIG. **3** illustrates an embodiment of the compressor unit **300** that processes chunks **302** or different words of a cache line, such as eight **32** byte words of a **256** byte cache line. The compressor unit **300** is comprised of data transformers **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4, including circuitry to perform different data transforms on the input chunk **302**, and one

passthrough unit **306** having circuitry to output the received input word **302** without any transformation. Data transformers **304**.sub.i encode data units of the chunk **302** to increase the number of zeroes in the transformed data units. The data units of the chunk **302** may comprise a bit or one or more bytes. The different data transformers **304**.sub.i may implement different transform methods or comprise the same method but with different parameters and tuning. For instance, the different data transformers **304**.sub.i may use the same transform method but process different item size data units of the input chunk **302**. For instance, as shown in FIG. **3**, the different item size of the data units processed by the data transformers **304**.sub.i include 8 bytes, 16 bytes, 32 bytes, and 64 bytes.
[0024] In one embodiment, the data transformers **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 each implement a combination of a delta, XOR, and bit plane transform, or “DXB”. The delta transform may subtract item size bytes in the input word **302** by subtracting all from a single base value or subtracting neighbor values. The XOR transform may XOR the delta transformed bytes, and the bit plane operation may then perform a bit plane transform on the data. In certain embodiments, the difference in the DXB transformers **304**.sub.i is that they each operate on different data unit sizes, e.g., 8, 16, 32, 64 bytes, of the input chunk **302**. In alternative embodiments, different data transforms may be used, and a data transformer **304**.sub.i may perform only one data transform or a different number and/or type of data transform than the DXB transform combination. The data transformation may involve delta and two-dimensional encoding. In the delta encoding, neighboring data items are subtracted from one another. In two-dimensional encoding, the encoding is performed both vertically (bit-plane) and horizontally (word-plane) and the best of the two dimensions producing the greatest number of zeroes is selected. The data transformers **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 implement arithmetic/logic
[0025] operations to transform the data units of the input chunk **302** to a transformed chunk encoded with more zeroes than the input chunk **302**. The different transform units **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 may compete in a tournament so the transformed data is selected that results in the minimum number of non-zero-bits.
[0026] Each population count unit **308**.sub.1, **308**.sub.2, **308**.sub.3, **308**.sub.4 comprises circuitry coupled to each data transformer **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 and a population count (“POPCOUNT”) unit **310** is coupled to the passthrough unit **306**. The population count units **308**.sub.1, **308**.sub.2, **308**.sub.3, **308**.sub.4 and **310** count the number of non-zero (NZ) bits in the output from the data transformers **304**.sub.1, **304**.sub.2, **3043**, **304**.sub.4 and the passthrough unit **306**, respectively. The non-zero bits from the population count units **308**.sub.1, **308**.sub.2, **308**.sub.3, **308**.sub.4, and **310** are inputted to a population count (POPCOUNT) decision unit **312**, comprising circuitry, that indicates the data transformer **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 or passthrough unit **306** outputting the transformed chunk having a fewest number of non-zero bits. This indication of the data transformer **304**.sub.i or passthrough unit **306** producing the fewest number of non-zero bits is provided as input to the select minimum output MUX **314**, which outputs the transformed chunk **316**, having the same byte size as the input word **302**, but encoded with a greater number of zeros.
[0027] The zero-value compressor **318** compresses the transformed chunk **316**, or input chunk **302**, to produce the compressed chunk **322** and a non-zero mask (NZMASK) **320** having bits indicating the non-zero values in the transformed chunk **316**. Packing logic **324** creates a compressed chunk package **400**, as shown in FIG. **4**, including a header **500** having metadata on the compressed chunk **322**, the non-zero mask (NZMASK) **320**, the compressed word **322**, and an error correction code (ECC) **402** calculated from an XOR operation on the header **500**, NZMASK **320**, and compressed word **322**. The packing logic **324** may generate a compressed chunk package **400**.sub.i for each of the n input words **302** processed for a cache line and store all the compressed chunk packages for the cache line in the cache memory cell array **202** to form a compressed cache line **326**.

[0028] FIG. **5** illustrates an embodiment of the header **500** in the compressed chunk package **400**.sub.i, and includes a bit plane encoding used flag **502** indicating whether the selected data transformer **304**; that outputs the data having the minimum number of non-zero bits performs a bit plane encoding transform; a data transform item size **504** indicating an item size of the data units the data transformer **304**; transforms in the input data chunk **302**; and a zero-value compression item size **506** indicating a size of the data units in the transformed chunk **316** the zero-value compressor **318** processes to remove data units having a zero value, such as a number of bits or bytes having a zero value.

[0029] FIG. **6** illustrates an embodiment of operations performed by the compressor unit **300** to compress an input data chunk **302**, such as a word, of a cache line to store in the cache memory **200**. Upon receiving (at block **600**) a cache line to compress, the compressor unit **300** performs a loop of operations **602** through **622** for each input data chunk **302**, e.g., 32 byte word, of the cache line to compress. The input data chunk **302** may comprise a word or other sized data unit. The input chunk **302** is inputted (at block **604**) to each of the data transformers **304**.sub.1, **304**.sub.2, **304**.sub.3, **304**.sub.4 and the passthrough unit **306**. Each data transformer **304**.sub.i performs (at block **606**) a different data transform on the input chunk **302**. In one embodiment, the data transformers **304**.sub.i may perform the same type of transform, e.g., DXB, but perform the transform on different item size data units of the input word **302**. The data transformer **304**.sub.i may increase the number of zeroes in the input chunk **302**. Each POPCOUNT unit **308**.sub.i and **310**, coupled to one of the data transformers **304**; and passthrough unit **306**, determines (at block **608**), for the transformed chunk, the number of non-zero bits in the transformed chunk or the input chunk **302** from the passthrough unit **306**.

[0030] The POPCOUNT decision unit **312** determines (at block **610**) the data transformer **304**.sub.i or the passthrough unit **306** that produces a transformed chunk or input data having the minimum number of non-zero bits and inputs to the MUX **314** indication of the transformer **304**.sub.i or passthrough unit **306** producing the minimum number of non-zero bits. The MUX **314** selects (at block **612**) the transformed chunk **316** or input chunk **302** having the minimum number of non-zero bits to output as the transformed chunk **316** to the zero-value compressor **318**. The zero-value compressor **318** removes (at block **614**) non-zero data units of a ZVC item size (e.g., 1 bit, 1 byte, 2 bytes, 4 bytes, 8 bytes, 16 bytes) to output the compressed chunk **322** and a non-zero bitmask (NZMASK) **320** indicating data units in the transformed chunk **316** having non-zero values.

[0031] Packing logic **324** generates (at block **616**) a header **500** for the compressed chunk **322** indicating whether bit plane encoding was used **502** in the data transform, data transform item size **504** of data units in the input chunk **302** transformed, and the zero-value compression item size **506**. The ZVC item size **506** may be provided for embodiments using an ensemble of zero-value compressors as described with respect to FIGS. **7**, **8**, and **9**, indicating an item size of the data unit subject to removal if the data unit has a non-zero value. The packing logic **324** further generates (at block **618**) an error correction code (ECC) **402** for the compressed chunk **322**, header **500**, and non-zero bitmask **320** and forms a compressed chunk package **400**.sub.i having the header **500**, non-zero bit mask **320**, compressed chunk **322**, and ECC code **402**. Control then proceeds (at block **620**) back to block **602** to process the next ith chunk **302** until all the chunks in the cache line are compressed. All the generated compressed chunk packages **400**.sub.1 . . . **400**.sub.n for the cache line are then written (at block **622**) to the cache memory cell array **202**.

[0032] Described embodiments optimize zero-value compression by having an ensemble of data transformers encode input chunks or words of a cache line with mostly zeroes to produce transformed chunks, and then select a transformed chunk having a minimum number of non-zero values to optimize the zero-value compressor **318**, which compresses by removing zero-values. Further, the data transformers **304**.sub.i transform the input chunk **302** in parallel on same clock cycles, so that the output having the minimum number of non-zero-values, i.e., most zero-values, is selected from transformed data from all the different transforms. In this way, the best transformed

data is selected to compress, which may change for different input chunks words, on the same clock cycle.
[0033] FIG. **7** illustrates an embodiment where the zero-value compressor **318** in FIG. **3** is implemented as a zero-value compressor **700** having an ensemble of different zero-value compressors **702**.sub.1, **702**.sub.2, **702**.sub.3, **702**.sub.4, **702**.sub.5, **702**.sub.6 that each receive a transformed chunk **704**, such as transformed chunk **316**. Although six zero value compressors **702**.sub.i
[0034] are shown there may be any number of multiple zero value compressors **702**.sub.i. Each of the ZVC compressors **702**.sub.i may process data units of different item sizes, e.g., 4 bits and 1, 2, 4, 8, and 16 bytes, such that a data unit having a zero value, i.e., all zero bits, is removed. For instance, a data unit of a transformed chunk **316** may have a 1 byte data unit of a zero value, but that same 1 byte in a 2 byte data unit may have a non-zero value if there are non-zero bytes in the 2 byte data unit. Each ZVC **702**.sub.i outputs a compressed chunk and a non-zero mask indicating non-zero data units.
[0035] A ZVC selector **706** selects a compressed chunk **708** outputted by one of the ZVCs **702**.sub.i that has a minimum number of non-zero bits and outputs a non-zero mask (NZMASK) **710** indicating data units of the ZVC item size having non-zero values, and a ZVC item size **712** of the data units subject to zero-value compression. For instance, if the input chunk **302** is 32 bytes and the ZVC item size is 16 bytes, then the NZMASK **710** has two bits, one for each of the 16 byte data units in the compressed chunk **708**. The non-zero bitmask **710**, compressed chunk **708**, and ZVC item size **712** may be sent to the packing logic **324** in FIG. **3** to be included in a compressed chunk package **400**.sub.i
[0036] and written to the cache memory cell array **202**.
[0037] FIG. **8** illustrates an embodiment of operations performed by the ZVC **700** of FIG. **7** to select a compressed chunk **708** from an ensemble of ZVCs **702**.sub.i having the minimum number of non-zero bits to replace the operation in block **614** in FIG. **6**. Upon receiving (at block **800**) a transformed chunk **316**, the transformed chunk **316** is inputted to each of the ZVCs **702**.sub.i. Each ZVC **702**.sub.i removes (at block **804**) zero value data units, of the ZVC item size for the ZVC **702**.sub.i, in the transformed chunk **316** to produce a compressed chunk having only non-zero data units, of the ZVC item size, and a non-zero mask indicating data units, of the ZVC item size, in the transformed chunk having non-zero values. Each ZVC **702**.sub.i may remove chunks for data units of different item sizes. The ZVC selector **706** selects (at block **806**) an output compressed chunk **708** (i.e., non-zero data units) and non-zero mask **710** for the compressed chunk **708** totaling a minimum number of bytes of the output compressed data and non-zero mask for all the ZVCs **702**.sub.i. Control then proceeds (at block **808**) to block **616** in FIG. **6** to continue processing of the compressed chunk **708**.
[0038] With the embodiment of FIG. **8**, an additional dimension of optimization is performed to have multiple ZVCs **702**.sub.i concurrently compress the transformed chunk for different size data units to determine the ZVC **702**.sub.i producing the compressed chunk **708** and non-zero bitmask **710** having a minimum number of bits to further optimize the compression to pick a best zero-value compression.
[0039] FIG. **9** illustrates how using different item sizes resulting in different size data units for the ZVCs **702**; produces different size non-zero bitmasks and non-zero values. Section **900** shows how different ZVC item sizes results in different compressed outputs for original data **902**. For instances of the four choices **900** of different data unit item sizes of 1, 2, 4, and 8 bytes, the ZVC **702**.sub.i removing data units of a one-byte item size produces the smallest compressed chunk **904** of non-zero bits, as compared to non-zero items **906**, **908**, **910**, but has the largest size non-zero bitmask **912** of non-zero bitmasks **914**, **916**, **918**. However, the “choice 1” of a one-byte data unit size results in the compressed chunk **904** and non-zero bitmask **912** having the smallest number bytes of the other choices for 2, 4, and 8 byte item sizes of the data units. The second section **920** shows

zero-value compression of 1 byte or 8 byte data unit item sizes. The 8 byte data unit size produces a non-zero bitmask **922** having significantly fewer bytes than non-zero bitmask **924** for the 1-byte item size, with compressed data **926** and **928**, respectively, having a same number of bytes. Thus, for the initial original data shown in section **920**, of the two options in section **920**, a zero-value compressor removing 8 byte size zero-value data units results in better compression than using a 1-byte item size data unit, or 10 bytes versus 8.25 bytes.
[0040] In certain instances, the item size **504** selected for the data transformer method **304**.sub.i may be different than the item size **506** selected for the zero-value compressor.
[0041] FIG. **10** illustrates an embodiment of a decompressor unit **1000**, such as the decompressor unit **1000** in FIG. **2**, that processes compressed encoded chunks **322** produced by the compressor unit **300**, such as eight 32 byte words of a 256 byte cache line. The decompressor unit **1000** is comprised of a ZVC decompressor **1002** to decompress the compressed encoded chunk **322** stored in the memory cell array **202** in a compressed chunk package **400**.sub.i. The ZVC decompressor **1002** uses the header **500** to identify the ZVC item size **506** used in compressed encoded chunk **322** and the ZVC NZMASK **318**, **710** for the compressed encoded chunk **322** to populate the compressed encoded chunk **322** with zero values at locations indicated in the mask **318** to produce a decompressed encoded chunk **1004** with the header **500**. The header **500** additionally identifies the data transform item size **504** used in the DXB encoded chunk and whether bit-plane encoding **502** is to be used or not in the decoders **1008**.sub.i. A decoder selector **1006** determines from the header **500** the data transform item size **504** or other indicator to determine the data decoder **1008**.sub.1, **1008**.sub.2, **1008**.sub.3 . . . **1008**.sub.4 or the passthrough circuit **1010** to which the decompressed encoded chunk **1004** is forwarded to decode. The passthrough unit **1010** forwards the decompressed chunk **1004** without any data decoding because during compression, the data chunk **302** was not subject to decoding and instead un-encoded through the passthrough unit **306** (FIG. **3**). The selected decoder **1008**; decodes the data to its original chunk **1012**, or chunk **302** in FIG. **3** before compression. The decoders **1008**.sub.1 . . . **1008**.sub.4 include circuitry to perform different decoding
[0042] operations to decode for different encoding techniques used to encode the data with more zeroes. The decoders **1008**.sub.i decode the encoded chunk **1004** to transform the chunk **1004** back to the original source data. The data units of the encoded chunk **1004** may comprise a bit or one or more bytes. The different data decoders **1008**.sub.i may implement different decoding methods or comprise the same decoding method but with different parameters and tuning. For instance, the different decoders **1008**.sub.i may use the same decoding method but process different item size data units of the encoded chunk **1004**. For instance, as shown in FIG. **10**, the different item size of the data units processed by the decoders **1008**.sub.i include 8 bytes, 16 bytes, 32 bytes, and 64 bytes.
[0043] In one embodiment, the decoders **1008**.sub.i each implement a combination of a delta decoder **1014**.sub.i, XOR decoder, and bit plane transform decoder, or “DXB” decoder, which would decode in the order of bit plane transform, XOR, and delta because, in the embodiment of FIG. **3**, the encoding was in the order of delta, XOR and bit plane transform. In certain embodiments, the difference in the DXB decoders **1008**.sub.i is that they each operate on different data unit sizes, e.g., 8, 16, 32, 64 bytes, of the encoded chunk **1004**. In alternative embodiments, different decoders transforms may be used, and a data decoder **1008**.sub.i may perform only one decoding operation or a different number and/or type of decoding than the DXB decoding combination. The data transformation may involve delta and two-dimensional encoding.
[0044] The decoders **1008**.sub.1, **1008**.sub.2, **1008**.sub.3, **1008**.sub.4 implement arithmetic/logic operations to decode the encoded chunk **1004** to the original chunk **1012**.
[0045] FIG. **11** illustrates an embodiment of a delta decoder **1100** for the 8 byte chunk size delta decoder **10141**. In FIG. **11**, the initial source data A**0**. . . . A**7** and encoded data B**0** . . . B**7** comprise 8 bytes of data. FIG. **11** shows how the encoded data B**0** . . . B**7** was created by subtracting each of

the eight bytes A**1**. . . . A**7** by a neighboring byte A**0**. . . . A**6** to the left in the word **302** of A**0** . . . A**7** bytes. The delta decoder **1100** of FIG. **11** uses the encoded input B**0** . . . B**7**, shown as **1200**.sub.1 in FIG. **12**, to reconstruct the original source A**0** . . . A**7** at the result output **1202**, shown in FIG. **12**. The delta decoder **1100** for a word size of eight bytes has three stages **1102**.sub.1, **1102**.sub.2, **1102**.sub.3. Each stage **1102**.sub.i includes at least one passthrough unit **1104**.sub.1, **1104**.sub.2 . . . **1104**.sub.7 and a row of adders **1106**.sub.1, **1106**.sub.2, **1106**.sub.3following passthrough units **1104**.sub.1, **1104**.sub.3, **1104**.sub.7, respectively, which in parallel process decoded data at different stages on the same clock cycle. The output of the stages **1102**.sub.i includes one or more recovered source values A.sub.j or the results of the sum of the inputs to the adders, which comprises a subtraction of two source values Ai-Aj.
[0046] FIG. **12** illustrates how the source values Ai are combined in each stage at the adders in parallel. In the first stage **1200**.sub.1 the encoded data B**0** . . . B**7** is shown as a function of the original source values, where each encoded item comprises the subtraction of two source values Ai-A(i-**1**). In the first stage **1200**.sub.1, each encoded item B**0** . . . B**6** is added, in parallel, to a neighboring encoded item B**1** . . . B**7** one hop to the right, respectively. In the second stage **1200**.sub.2, each first stage decoded item B**0**′ . . . B**5**′, resulting from the first stage **1200**.sub.1 decoding, which can be expressed as the below source values, is added, in parallel, to a neighboring first stage encoded item B**2**′ . . . B**7**′ two hops to the right, respectively. The first stage decoded items B**0**′ and B**1**′ comprise the recovered source values A**0** and A**1**, respectively, which are forwarded to the passthrough units **1104**.sub.2, **1104**.sub.3. In the third stage **1200**.sub.3, each second stage decoded item B**0**″ . . . B**3**″, resulting from the second stage decoding **1200**.sub.2, which can be expressed as the below source values, is added to a neighboring second stage encoded item B**4**″ . . . B**7**″ four hops to the right, respectively. The second stage decoded items B**0**″ . . . B**3**″ comprise the recovered source values A**1** . . . A**3**, respectively, which are forwarded to passthrough units **1104**.sub.4 . . . **1104**.sub.7, respectively. The output of the third stage decoding **1200**.sub.3 comprises the final decoded source values **1202**, or A**0** . . . A**7**.
[0047] The circuit structure can be applied to words having any number of units, that are powers of two, such as more than 8 bytes, or 4, 16, 32, 64, et seq. bytes. If the number of bytes or data units comprises n, then there would be log.sub.2(n) stages. Further, at each stage i, for i=0 . . . (n−1), there are **2**.sup.i passthrough units followed by (n−2.sub.i) adders. The first **2**.sup.i inputs to the ith stage are forwarded to passthrough units and the first **2**.sup.i adders to be combined with inputs (2.sup.i+1) through (n−1), respectively.
[0048] FIG. **13** illustrates an embodiment of operations performed by a delta decoder having log.sub.2(.sub.n) stages to decode a delta encoded word having n items. The encoded item B.sup.ij refers to output from an ith stage for decoded item j, where i=0 refers to the decoded word before subject to the stages of adders. Upon the delta decoder **1014**.sub.i receiving (at block **1300**) an encoded word of n items at a log.sub.2(n) stage delta decoder, for the first stage, in parallel the following operations (at block **1302**) are simultaneously performed: input the first encoded item B.sup.00 to the passthrough unit; input encoded items B.sup.01 . . . B.sup.0n−1 to adders 1 to (n−1) respectively, and input encoded items B.sup.00 . . . B.sup.0n−2 to adders 1 . . . n−1, respectively, one hop away, resulting in adding B.sup.0j+B.sup.0k for j=0 . . . n−2 and k=1 . . . n−1. For each remaining ith stage of log.sub.2(n) stages, where i=1 to log.sub.2(n), in parallel process n output items from previous stage (at blocks **1304** through **1312**) as follows: input (at block **1306**) items 1 to 2.sup.i (B.sup.i0 . . . B.sup.i2.sup.i) from previous stage, to passthrough circuits 1 to 2.sup.i; input (at block **1308**) items 1 to (n−2.sup.i) (B.sup.i 0 . . . B.sup.i (n−2.sup.i)), from previous stage, to adders 1 to (n−2.sup.i), **2**.sup.i hops away; input (at block **1310**) items (1+2.sup.i) to n (B.sup.i 2.sup.i . . . B.sup.i n), from previous stage, to adders 1 to n−2.sup.1, 2.sup.i hops away These operations for the ith stage result in adding B.sup.ij+B.sup.ik for j=0 . . . (2.sup.i−1) and k=2.sup.i . . . n−1. The output from the last stage, log.sub.2(n) stage, comprises the final decoded output, or A**0** . . . An.

[0049] With the above embodiments, the decoding of delta encoded data is optimized by performing in parallel log.sub.2(n) stages of operations in parallel to reduce the latency over delta decoding techniques that decode data sequentially for each item, requiring n sequential operations. In this way described embodiments, optimize and reduce the latency of delta decoding operations using multiple stages of adders to perform stages of decoding operations in parallel on substantially fewer clock cycles than needed to sequentially decode each delta encoded item.
[0050] The letter designators, such as i, n, among others, are used to designate an instance of an element, i.e., a given element, or a variable number of instances of that element when used with the same or different elements.
[0051] The terms “an embodiment”, “embodiment”, “embodiments”, “the embodiment”,
[0052] “the embodiments”, “one or more embodiments”, “some embodiments”, and “one embodiment” mean “one or more (but not all) embodiments of the present invention(s)” unless expressly specified otherwise.
[0053] The terms “including”, “comprising”, “having” and variations thereof mean “including but not limited to”, unless expressly specified otherwise.
[0054] The enumerated listing of items does not imply that any or all the items are mutually exclusive, unless expressly specified otherwise.
[0055] The terms “a”, “an” and “the” mean “one or more”, unless expressly specified otherwise.
[0056] Devices that are in communication with each other need not be in continuous communication with each other, unless expressly specified otherwise. In addition, devices that are in communication with each other may communicate directly or indirectly through one or more intermediaries.
[0057] A description of an embodiment with several components in communication with each other does not imply that all such components are required. On the contrary a variety of optional components are described to illustrate the wide variety of possible embodiments of the present invention.
[0058] When a single device or article is described herein, it will be readily apparent that more than one device/article (whether or not they cooperate) may be used in place of a single device/article. Similarly, where more than one device or article is described herein (whether or not they cooperate), it will be readily apparent that a single device/article may be used in place of the more than one device or article or a different number of devices/articles may be used instead of the shown number of devices or programs. The functionality and/or the features of a device may be alternatively embodied by one or more other devices which are not explicitly described as having such functionality/features. Thus, other embodiments of the present invention need not include the device itself.
[0059] The foregoing description of various embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims herein after appended.

Claims

1. A decoder unit implemented in a cache memory having a cache memory cell array, comprising: a first stage of adder circuits to add, in parallel, pairs of encoded items transformed using a delta encoding, wherein the encoded items include a plurality of deltas of neighbors of sequential source items; and at least one successive stage of adder circuits to add, in parallel in each stage, pairs of

outputs from a previous stage of adder circuits, wherein each successive stage has fewer adder circuits than the previous stage, and wherein output from a last of the at least one successive stage comprises the sequential source items.
2. The decoder unit of claim 1, wherein the adding, in parallel, pairs of encoded items in the first stage of adder circuits comprises adding, in parallel, each pair of consecutive decoded items to produce first output items, wherein the adding, in parallel, pairs in each successive stage of adder circuits comprises adding items multiple hops away to produce fewer added outputs than at the previous stage of adder circuits.
3. The decoder unit of claim 1, wherein each of the sequential source items comprises a number of bytes of data to encode.
4. The decoder unit of claim 1, wherein there are n encoded items, wherein the first stage of adder circuits comprises n-**1** adder circuits to add each consecutive pair of the encoded items to produce n outputs including n-**1** added outputs, wherein the at least one successive stage of adder circuits comprises: at least one successive ith stage of adders following the first stage, where i=1 . . . j, and wherein each successive stage adds a pair of outputs from a previous stage 2.sup.i hops away.
5. The decoder unit of claim 4, wherein each successive ith stage following the first stage has (n−2.sup.i) adders.
6. The decoder unit of claim 4, wherein j =log.sub.2(n).
7. The decoder unit of claim 4, wherein each stage outputs n items, wherein the first stage includes one passthrough circuit followed by the n−1 adder circuits, wherein the passthrough circuit passes a first input item to first stage output, also including output from the n−1 adder circuits, to a second stage of adder circuits, wherein each successive ith stage following the first stage has (n−2.sup.i) adders following 2.sup.i passthrough circuits to process output from a previous stage.
8. The decoder unit of claim 4, wherein the encoded items are provided by a decompressor unit in a cache memory controller of the cache memory from a compressed word in a compressed cache line in the cache memory cell array.
9. A decompressor unit implemented in a cache memory having a cache memory cell array, comprising: a decompressor to decompress a compressed chunk of encoded items stored in the cache memory cell array to produce an uncompressed encoded chunk of the encoded items, wherein the encoded items are formed using delta encoding and include a plurality of deltas of neighbors of sequential source items; and a decoder including: a first stage of adder circuits to add, in parallel, pairs of the encoded items; and at least one successive stage of adder circuits to add, in parallel in each stage, pairs of outputs from a previous stage of adder circuits, wherein each successive stage has fewer adder circuits than the previous stage, and wherein output from a last of the at least one successive stage comprises the sequential source items.
10. The decompressor unit of claim 9, wherein the adding, in parallel, pairs of encoded items in the first stage of adder circuits comprises adding, in parallel, each pair of consecutive decoded items to produce first output items, wherein the adding, in parallel, pairs in each successive stage of adder circuits comprises adding items multiple hops away to produce fewer added outputs than at the previous stage of adder circuits.
11. The decompressor unit of claim 9, wherein there are n encoded items, wherein the first stage of adder circuits comprises n−1 adder circuits to add each consecutive pair of the encoded items to produce n outputs including n−1 added outputs, wherein the at least one successive stage of adder circuits comprises: at least one successive ith stage of adders following the first stage, where i=1 . . . j, and wherein each successive stage adds a pair of outputs from a previous stage 2.sup.i hops away.
12. The decompressor unit of claim 11, wherein each successive ith stage following the first stage has (n−2.sup.i) adders.
13. The decompressor unit of claim 11, wherein j =log.sub.2(n).
14. The decompressor unit of claim 11, wherein each stage outputs n items, wherein the first stage

includes one passthrough circuit followed by the n−1 adder circuits, wherein the passthrough circuit passes a first input item to first stage output, also including output from the n−1 adder circuits, to a second stage of adder circuits, wherein each successive ith stage following the first stage has (n−2.sup.i) adders following 2.sup.i passthrough circuits to process output from a previous stage.
15. A method for decoding data in a cache memory having a cache memory cell array, comprising: adding, by a first stage of adder circuits, in parallel, pairs of encoded items transformed using a delta encoding, wherein the encoded items include a plurality of deltas of neighbors of sequential source items; and adding, by at least one successive stage of adder circuits, in parallel in each stage, pairs of outputs from a previous stage of adder circuits, wherein each successive stage has fewer adder circuits than the previous stage, and wherein output from a last of the at least one successive stage comprises the sequential source items.
16. The method of claim 15, wherein the adding, in parallel, pairs of encoded items in the first stage of adder circuits comprises adding, in parallel, each pair of consecutive decoded items to produce first output items, wherein the adding, in parallel, pairs in each successive stage of adder circuits comprises adding items multiple hops away to produce fewer added outputs than at the previous stage of adder circuits.
17. The method of claim 15, wherein there are n encoded items, wherein the first stage of adder circuits comprises n−1 adder circuits to add each consecutive pair of the encoded items to produce n outputs including n−1 added outputs, wherein the at least one successive stage of adder circuits comprises: at least one successive ith stage of adders following the first stage, where i=1 . . . j, and wherein each successive stage adds a pair of outputs from a previous stage 2.sup.i hops away.
18. The method of claim 17, wherein each successive ith stage following the first stage has (n−2.sup.i) adders.
19. The method of claim 17, wherein j=log.sub.2(n).
20. The method of claim 15, further comprising: decompressing, by a decompressor, a compressed chunk of encoded items stored in the cache memory cell array to produce an uncompressed encoded chunk of the encoded items, wherein the encoded items are formed using delta encoding and include a plurality of deltas of neighbors of sequential source items, and wherein the encoded chunk is provided to the first stage of adder circuits to process.