Bitmap Indices for Data Warehouses

Kurt Stockinger and Kesheng Wu
Computational Research Division
Lawrence Berkeley National Laboratory
University of California
Mail Stop 50B-3238
1 Cyclotron Road
Berkeley, California 94720, USA
{KStockinger, KWu}@lbl.gov

Tel: +1 (510) 486 5519
Fax: +1 (510) 486 4004

BITMAP INDICES FOR DATA WAREHOUSES

ABSTRACT

In this chapter we discuss various bitmap index technologies for efficient query processing in data warehousing applications. We review the existing literature and organize the technology into three categories, namely bitmap ***encoding***, ***compression*** and ***binning***. We introduce an efficient bitmap **compression** algorithm and examine the space and time complexity of the compressed bitmap index on large data sets from real applications. According to the conventional wisdom, bitmap indices are only efficient for low-cardinality attributes. However, we show that the compressed bitmap indices are also efficient for high-cardinality attributes. Timing results demonstrate that the bitmap indices significantly outperform the projection index, which is often considered to be the most efficient access method for multi-dimensional queries. Finally, we review the bitmap index technology currently supported by commonly used commercial database systems and discuss open issues for future research and development.

KEYWORDS

Bitmap index, compression, query processing, performance evaluation

INTRODUCTION

Querying large data sets to locate some selected records is a common task in data warehousing applications. However, answering these queries efficiently is often difficult due to the complex nature of both the data and the queries. The most straightforward way of evaluating a query is to sequentially scan all data records to determine whether each record satisfies the specified conditions. A typical query condition is as follows "Count the number of cars sold by producer *P* in the time interval *T*". This search procedure could usually be accelerated by *indices*, such as variations of *B-Trees* or *kd-Trees* (Comer, 1979; Gaede & Guenther, 1998). Generally, as the number of attributes in a data set increases, the number of possible indexing combinations increases as well. To answer multi-dimensional queries efficiently, one faces a difficult choice. One possibility is to construct a separate index for each combination of attributes, which requires an impractical amount of space. Another possibility is to choose one of the multi-dimensional indices, which is only efficient for some of the queries. In the literature, this dilemma is often referred to as the *curse of dimensionality* (Berchtold et al., 1998; Keim et al., 1999).

In this chapter we discuss an indexing technology that holds a great promise in breaking the curse of dimensionality for data warehousing applications, namely the *bitmap index*. A very noticeable character of a bitmap index is that its primary solution to a query is a *bitmap*. One way to break the curse of dimensionality is to build a bitmap index for each attribute of the data set. To resolve a query involving conditions on *multiple attributes*, we first resolve the conditions on each attribute using the corresponding bitmap index, and obtain a solution for each condition as a bitmap. We then obtain the answer to the overall query by combining these bitmaps. Because the operations on bitmaps are well supported by computer hardware, the bitmaps can be combined easily and efficiently. Overall, we expect the total query response time to *scale linearly in the number of attributes* involved in the *query*, rather than exponentially in the number of dimensions (attributes) of the *data set*, thus breaking the curse of dimensionality.

The above statements omitted many technical details that we will elaborate in this chapter. In the next section we give a broad overview of the *bitmap index* and its relative strengths and

weaknesses to other common indexing methods. We then describe the *basic bitmap index* and define the terms used in the discussions. We devote a large portion of this chapter to review the three orthogonal sets of strategies to improve the basic bitmap index. After reviewing these strategies, we give a more in-depth discussion on how the *Word-Aligned-Hybrid* (WAH) bitmap compression technique reduces the bitmap index sizes. We will also present some timing results to demonstrate the effectiveness of the WAH compressed bitmap indices for two different application data sets. Our performance evaluation is deliberately based on data sets with *high-cardinality attributes*, since for low-cardinality attributes the performance advantage of bitmap indices is well known. We conclude with a short review of bitmap indices available in commercial DBMS products and discuss how to make bitmap indices better supported in these commercial products.

BACKGROUND

By far the most commonly used indexing method is the *B-Tree* (Comer, 1979). Almost every database product has a version thereof since it is very effective for *on-line transaction processing* (OLTP). This type of *tree-based indexing method* has nearly the same operational complexities for *searching* and *updating* the indices. This parity is important for OLTP because searching and updating are performed with nearly the same frequencies. However, for most data warehousing applications such as *on-line analytical processing* (OLAP), the searching operations are typically performed with a much higher frequency than that of updating operations (Chaudhuri, 1997; Chaudhuri, 2001). This suggests that the indexing methods for OLAP must put more emphasis on searching than on updating. Among the indexing methods known in the literature, the bitmap index has the best balance between searching and updating for OLAP operations.

Frequently, in OLAP operations each query involves a number of attributes. Furthermore, each new query often involves a different set of attributes than the previous one. Using a typical multi-dimensional indexing method, a separate index is required for nearly every combination of attributes (Gaede & Guenther, 1998). It is easy to see that the number of indices grows exponentially with the number of attributes in a data set. In the literature this is sometimes called the *curse of dimensionality* (Berchtold et al., 1998; Keim et al., 1999). For data sets with a moderate number of dimensions, a common way to cure this problem is to use one of the multi-dimensional indexing methods, such as *R-Trees* or *kd-trees*. These approaches have two notable shortcomings. Firstly, they are effective only for data sets with *modest number of dimensions*, say, < 15. Secondly, they are only efficient for *queries involving all indexed attributes*. However, in many applications only *some of the attributes* are used in the queries. In these cases, the conventional indexing methods are often not efficient. For *ad hoc range queries*, most of the known indexing methods do not perform better than the *projection index* (O'Neil & Quass, 1997), which can be viewed as one way to organize the base. The bitmap index, on the other hand, has excellent performance characteristics on these queries. As shown with both theoretical analyses and timing measurements, a compressed bitmap index can be very efficient in answering one-dimensional range queries (Stockinger et al., 2002; Wu et al., 2004; Wu et al., 2006). Since answers to one-dimensional range queries can be efficiently combined to answer arbitrary multi-dimensional range queries, compressed bitmap indices are efficient for any range query. In terms of computational complexity, one type of compressed bitmap index was shown to be theoretically optimal for one-dimensional range queries. The reason for the theoretically proven optimality is that the query response time is a linear function of the number of hits, i.e. the size of the result set. There are a number of indexing methods, including *B*-tree* and *B^+-tree*

(Comer, 1979), that are theoretically optimal for one-dimensional range queries, but most of them cannot be used to efficiently answer arbitrary multi-dimensional range queries.

The bitmap index in its various forms was used a long time before relational database systems or data warehousing systems were developed. Earlier on, the bitmap index was regarded as a special form of *inverted files* (Knuth, 1998). The *bit-transposed file* (Wong et al., 1985) is very close to the bitmap index currently in use. The name *bitmap index* was popularized by O'Neil and colleagues (O'Neil, 1987; O'Neil & Quass, 1997). Following the example set in the description of *Model 204*, the first commercial implementation of bitmap indices (O'Neil, 1987), many researchers describe bitmap indices as a variation of the B-tree index. To respect its earlier incarnation as inverted files, we regard a bitmap index as a data structure consisting of keys and bitmaps. Moreover, we regard the B-tree as a way to layout the keys and bitmaps in files. Since most commercial implementations of bitmap indices come after the product already contains an implementation of a B-tree, it is only natural for those products to take advantage of the existing B-tree software. For new developments and experimental or research codes, there is no need to couple a bitmap index with a B-tree. For example, in a research program that implements many of the bitmap indexing methods discussed later in this chapter (FastBit, 2005), the keys and the bitmaps are organized as simple arrays in a binary file. This arrangement was found to be more efficient than implementing bitmap indices in B-trees or as layers on top of a DBMS (Stockinger et al. 2002; Wu et al. 2002).

The *basic bitmap index* uses each distinct value of the indexed attribute as a key, and generates one bitmap containing as many bits as the number of records in the data set for each key (O'Neil, 1987). Let the *attribute cardinality* be the number of distinct values present in a data set. The size of a basic bitmap index is relatively small for low-cardinality attributes, such as "gender," "types of cars sold per month," or "airplane models produced by Airbus and Boeing." However, for high-cardinality attributes such as "temperature values in a supernova explosion," the index sizes may be too large to be of any practical use. In the literature, there are three basic strategies to reduce the sizes of bitmap indices: (1) using more complex bitmap ***encoding*** methods to reduce the number of bitmaps or improve query efficiency, (2) *compressing* each individual bitmap, and (3) using ***binning*** or other mapping strategies to reduce the number of keys. In the remaining discussions, we refer to these three strategies as encoding, compression and binning, for short.

BITMAP INDEX DESIGN

Basic Bitmap Index

Bitmap indices are one of the most efficient indexing methods available for speeding up multi-dimensional range queries for read-only or read-mostly data (O'Neil, 1987; Rotem et al., 2005b; Wu et al., 2006). The queries are evaluated with bitwise logical operations that are well supported by computer hardware. For an attribute with c distinct values, the basic bitmap index generates c bitmaps with N bits each, where N is the number of records (rows) in the data set. Each bit in a bitmap is set to "1" if the attribute in the record is

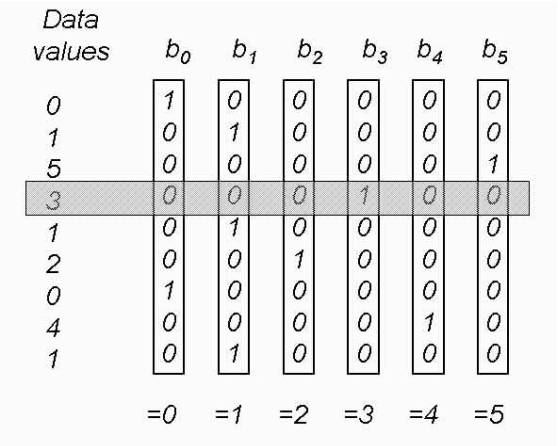

Figure 1: Simple bitmap index with 6 bitmaps to represent 6 distinct attribute values.

of a specific value; otherwise the bit is set to "0". Figure 1 shows a simple bitmap index with 6 bitmaps. Each bitmap represents a distinct attribute value. For instance, the attribute value 3 is highlighted to demonstrate the encoding. In this case, bitmap 3 is set to "1", all other bits on the same horizontal position are set to "0".

Encoding

The basic bitmap index introduced above is also called *equality-encoded* bitmap index since each bitmap indicates whether or not an attribute value equals to the key. This strategy is the most efficient for *equality queries* such as *"temperature = 100."* Chan and Ioannidis (1998; 1999) developed two other encoding strategies that are called *range encoding* and *interval encoding*. These bitmap indices are optimized for *one-sided* and *two-sided range queries*, respectively. An example of a one-sided range query is "*pressure < 56.7*". A two-sided range query, for instance, is "*35.8 < pressure < 56.7*".

A comparison of an equality-encoded and range-encoded bitmap index is given in Figure 2 (based on Chan & Ioannidis, 1999). Let us look at the encoding of value 2, which is highlighted in the figure. For *equality encoding*, the third bitmap is set to "1" (E^2), whereas all other bits on the same horizontal line are set to "0". For the *range-encoded* bitmap index, all bits between bitmap R^2 and R^8 are set to "1", the remaining bits are set to "0". This encoding is very efficient for evaluating range queries. Consider, for instance, the query "*A <= 4*". In this case, at most one bitmap, namely bitmap R^4, has to be accessed (scanned) for processing the query. All bits that are set to "1" in this bitmap fulfill the query constraint. On the other hand, for the *equality-encoded* bitmap index, the bitmaps E^0 to E^4 have to be *ORed* together (via the *Boolean operator OR*). This means that 5 bitmaps have to be accessed, as opposed to only 1 bitmap for the case of *range encoding*. In short, *range encoding* requires at most *one bitmap scan* for evaluating range queries, whereas *equality encoding* requires in the worst case *c/2* bitmap scans, where *c* corresponds to the number of bitmaps. Since one bitmap in *range encoding* contains only "1"s, this bitmap is usually not stored. Therefore, there are only *c-1* bitmaps in a range-encoded index.

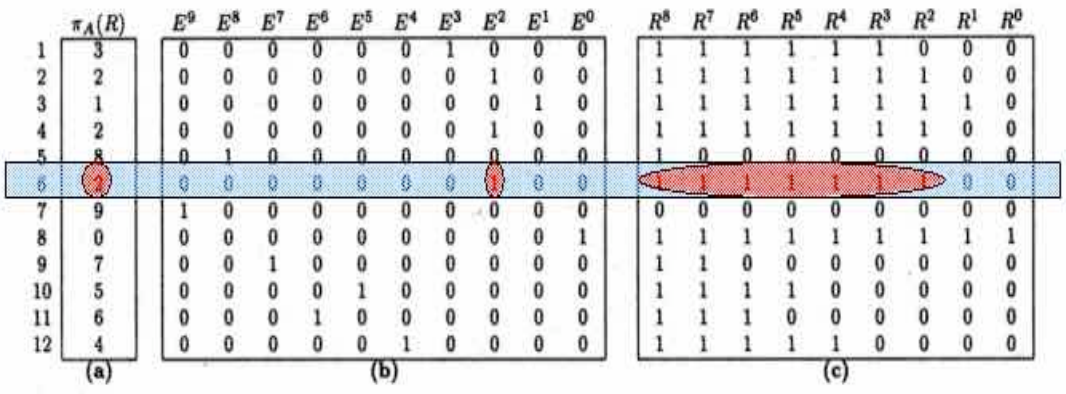

Figure 2: Equality-encoded bitmap index (b) compared with range-encoded bitmap index (c). The leftmost column shows the row ids (RID) for the data values represented by the projection index shown in (a).

Assuming each attribute value fits in a 32-bit machine word, the basic bitmap index for an attribute with **cardinality** 32 takes as much space as the *base data* (known as user data or original data). Since a B-tree index for a 32-bit attribute is often observed to use 3 or 4 times the space as the base data, many users consider only attributes with cardinalities less than 100 to be

suitable for using bitmap indices. Clearly, controlling the size of the bitmap indices is crucial to make bitmap indices practically useful for **higher cardinality** attributes. The *interval-encoding* scheme (Chan & Ioannidis, 1999) reduces the number of bitmaps only by a factor 2. Thus, other techniques are needed to make bitmap indices practical for high cardinality attributes.

The encoding method that produces the least number of bitmaps is *binary encoding* introduced by Wong et al. (1985). *Binary encoding* was later used by various authors (O'Neil & Quass, 1997; Wu & Buchmann, 1998) in the context of bitmap indices. This encoding method uses only $\lceil \log_2 c \rceil$ rather than $c/2$ bitmaps, where c is the **attribute cardinality**. For an integer attribute in the range of 0 and c-1, each bitmap in the bitmap index is a concatenation of one of the $\lceil \log_2 c \rceil$ binary digits for every record. For an attribute with c=1000, it only needs 10 bitmaps. The advantage of this encoding is that it requires much fewer bitmaps than *interval encoding*. However, to answer a range query, using *interval encoding* one has to access only *two bitmaps* whereas using *binary encoding* one usually has to access *all bitmaps*.

A number of authors have proposed strategies to find the balance between the space and time requirements (Wong et al., 1985; Chan & Ioannidis, 1999). A method proposed by Chan & Ioannidis (1999) called *multi-component encoding* can be thought of as a generalization of *binary encoding*. In the binary encoding, each bitmap represents a binary digit of the attribute values; the multi-component encoding breaks the values in a more general way, where each component could have a different size. Consider an integer attribute with values ranging from 0 to c-1. Let b_1 and b_2 be the sizes of two components c_1 and c_2, *where* $b_1 * b_2 >= c$. Any value v can be expressed as $v = c_1 * b_2 + c_2$, where $c_1 = v / b_2$ and $c_2 = v \% b_2$, where '/' denotes the integer division and '%' denotes the modulus operation. One can use a simple bitmap encoding method to encode the values of c_1 and c_2 separately. Next, we give a more specific example to illustrate the multi-component encoding.

Figure 3 illustrates a 2-component encoded bitmap index for an attribute with cardinality c=1000. In our example, the two components have base sizes of b_1=25 and b_2=40. Assume the attribute values are in the domain of [0; 999]. An attribute value v is decomposed into two components with $c_1 = v / 40$ and $c_2 = v \% 40$. The component c_1 can be treated as an integer attribute in the range of 0 and 24; the component c_2 can be viewed as an integer attribute in the range of 0 and 39. Two bitmap indices can be built, one for each component, for example, c_1 with the equality encoding and c_2 with *range enco*ding. If *range encoding* is used for both components, it uses 24 bitmaps for Component 1 and 39 bitmaps for Component 2. In this case, the 2-component encoding uses 63 bitmaps, which is more than the 10 bitmaps used by *binary encoding*. To answer the same query "v < 105" using the 2-component index, the query is effectively translated to "c_1<2 OR (c_1=2 AND c_2<25)." Evaluating this expression requires three bitmaps representing "c_1<=1," "c_1<=2," and "c_2<=24." In contrast, using the binary encoded bitmap index to evaluate the same query, all 10 bitmaps are needed.

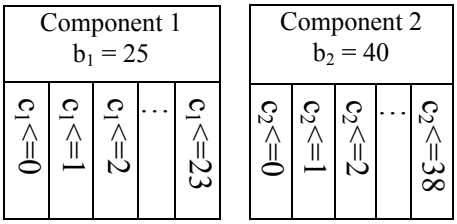

Figure 3: An illustration of a 2-component bitmap index.

Using more components can reduce the number of bitmaps and therefore reduces the total index size. However, using more components will also increase the number of bitmaps accessed in order to answer a query, hence increasing the query response time. Clearly, there is a trade-off

between the index size and the query response time. Without considering compression, Chan & Ioannidis (1999) have analyzed this space-time trade-off. They suggested that the *inflection point* of the *trade-off curve* is at 2 components. They further suggested that the two components should have nearly the same base sizes to reduce the index size.

Binning

The simplest form of bitmap indices works well for low-cardinality attributes, such as "gender," "types of cars sold per month," or "airplane models produced by Airbus and Boeing." However, for high-cardinality attributes such as "distinct temperature values in a supernova explosion," simple bitmap indices are impractical due to large storage and computational complexities. We have just discussed how different encoding methods could reduce the index size and improve query response time. Next, we describe a strategy called ***binning*** to reduce the number bitmaps. Since the encoding methods described before only take certain integer values as input, we may also view binning as a way to produce these integer values (bin numbers) for the encoding strategies.

The basic idea of binning is to build a bitmap for a *bin* rather than each *distinct attribute value*. This strategy disassociates the number of bitmaps from the attribute cardinality and allows one to build a bitmap index of a prescribed size, no matter how large the attribute cardinality is. A clear advantage of this approach is that it allows one to control the index size. However, it also introduces some uncertainty in the answers if one only uses the index. To generate precise answers, one may need to examine the original data records (candidates) to verify that the user-specified conditions are satisfied. The process of reading the base data to verify the query conditions is called ***candidate check*** (Stockinger et al., 2004; Rotem et al., 2005b).

A small example of an equality-encoded bitmap index with binning is given in Figure 4. In this example we assume that an attribute *A* has values between 0 and 100. The values of the attribute *A* are given in the second leftmost column. The range of possible values of *A* is partitioned into five bins [0, 20), [20, 40).... A "1-bit" indicates that the attribute value *falls into a specific bin*. On the contrary, a "0-bit" indicates that the attribute value *does not* fall into the specific bin. Take the example of evaluating the query "Count the number of rows where 37 <= A < 63". The correct result should be 2 (rows 5 and 7). We see that the range in the query overlaps with bins 1, 2 and 3. We know for sure that all rows that fall into bin 2 *definitely qualify* (i.e., they are *hits*). On the other hand, rows that fall into bins 1 and 3 *possibly qualify* and need further verification. In this case, we call bins 1 and 3 *edge bins*. The rows (records) that fall into edge bins are *candidates* and need to be checked against the query constraint.

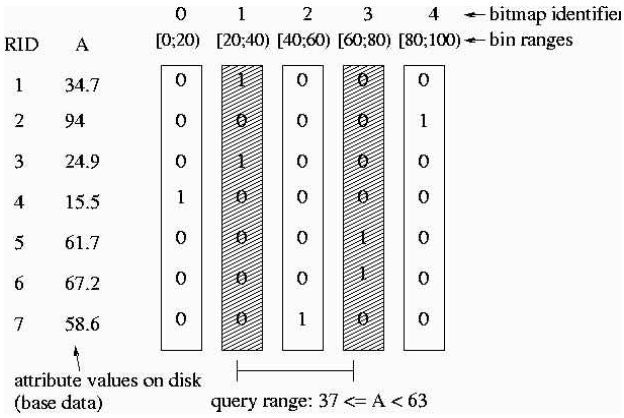

Figure 4: Range query "37 <= A < 63" on a bitmap index with binning.

In our example, there are four candidates, namely rows 1 and 3 from bin 1, and rows 5 and 6 from bin 3. The candidate check process needs to read these four rows from disk and examine their values to see whether or not they satisfy the user-specified conditions. On a large data set, a

candidate check may need to read many pages and may dominate the overall query response time (Rotem et al., 2005b).

There are a number of strategies to minimize the time required for the candidate check (Koudas, 2000; Stockinger et al., 2004; Rotem et al. 2005a, 2005b). Koudas (2000) considered the problem of finding the optimal binning for a given set of equality queries. Rotem et al. (2005a, 2005b) considered the problem of finding the optimal binning for range queries. Their approaches are based on dynamic programming. Since the time required by the dynamic programming grows quadratic with the problem size, these approaches are only efficient for attributes with relatively small attribute cardinalities (Koudas, 2000) or with relatively small sets of known queries (Stockinger et al. 2004). Stockinger et al. (2004) considered the problem of optimizing the order of evaluating multi-dimensional range queries. The key idea is to use more operations on bitmaps to reduce the number of candidates checked. This approach usually reduces the total query response time. Further improvements to this approach are to consider the attribute distribution and other factors that influence the actual time required for the candidate check.

To minimize number of disk page accesses during the candidate check, it is necessary to *cluste*r the attribute values. A commonly used clustering (*data layout*) technique is called the *vertical partition* or otherwise known as *projection index*. In general, the vertical data layout is more efficient for searching, while the horizontal organization (commonly used in DBMS) is more efficient for updating. To make the candidate check more efficient, we recommend the *vertical data organization*.

Compression

Compression is the third strategy to reduce the size of bitmap indices. Since each bitmap of the bitmap index may be used separately from others, compression is typically applied on each individual bitmap. Compression is a well-researched topic and efficient compression software packages are widely available. Even though these general-purpose compression methods are effective in reducing the size of bitmaps, query-processing operations on compressed bitmaps are often slower than on uncompressed bitmaps (Johnson, 1999). This motivated a number of researchers to improve the efficiency of compressed bitmap indices. Two of the most notable compression methods are Byte-aligned Bitmap Code (BBC) (Antoshenkov, 1994; Antoshenkov, 1996) and Word-Aligned Hybrid (WAH) code (Wu et al., 2004; Wu et al., 2006). Bitmaps compressed with BBC are slightly larger in size than those compressed with the best available general-purpose compression methods. However, operations on BBC compressed bitmaps are usually faster (Johnson, 1999). Clearly, there is a worthwhile space-time trade-off. The WAH compression takes this space-time trade-off one step further. More specifically, WAH compressed bitmaps are larger than BBC compressed ones, but operations on WAH compressed bitmaps are much faster than on BBC compressed ones. Therefore, WAH compressed bitmap indices can answer queries much faster as demonstrated in a number of different experiments (Stockinger et al. 2002; Wu et al., 2006). In the next section we provide a detailed description of the WAH compression. For more information on BBC, we refer the reader to (Antoshenkov, 1994; Antoshenkov, 1996).

WAH BITMAP COMPRESSION

The WAH bitmap compression is based on *run-length encoding*, where consecutive identical bits are represented with their *bit value* (0 or 1) and a *count* (length of the run). In WAH each such

run consists of a *fill* and a *tail*. A *fill* is a set of consecutive identical bits that is represented as a count plus their bit value. A *tail* is a set of mixed 0s and 1s that is represented literally without compression. One key idea of WAH is to define the *fills and tails* so that they can be *stored in words* - the smallest operational unit of modern computer hardware.

The WAH compression is illustrated in Figure 5. Assuming that a *machine word* is 32 bits long, the example shows how a sequence of 5456 bits (see Figure 5a)) is broken into two *runs* and encoded as three *words*. Conceptually, the bit sequence is first broken into groups of 31 bits each (see Figure 5b)). Next, the neighboring groups with identical bits are merged (Figure 5c)). Finally, these three groups are encoded as 32-bit machine words (Figure 5d)). The first run contains a fill of length 0 and a tail. There is no *fill word* but only a *literal word* representing the 31 tail bits for this run. Since a literal word has 32 bits, we use the first bit to indicate it is a literal word, and the rest to store the 31 tail bits. The second run contains a fill of length 174 (and thus represents 174 groups of 31 bits each) plus a tail. This run requires a *fill word* and a *tail word*. As illustrated, the first bit of a fill word indicates that it is a fill word, the second bit stores the bit value of the fill, which is 0 in this example. The remaining 30 bits store the binary version of the fill length, which is 10101110 (174) in this example.

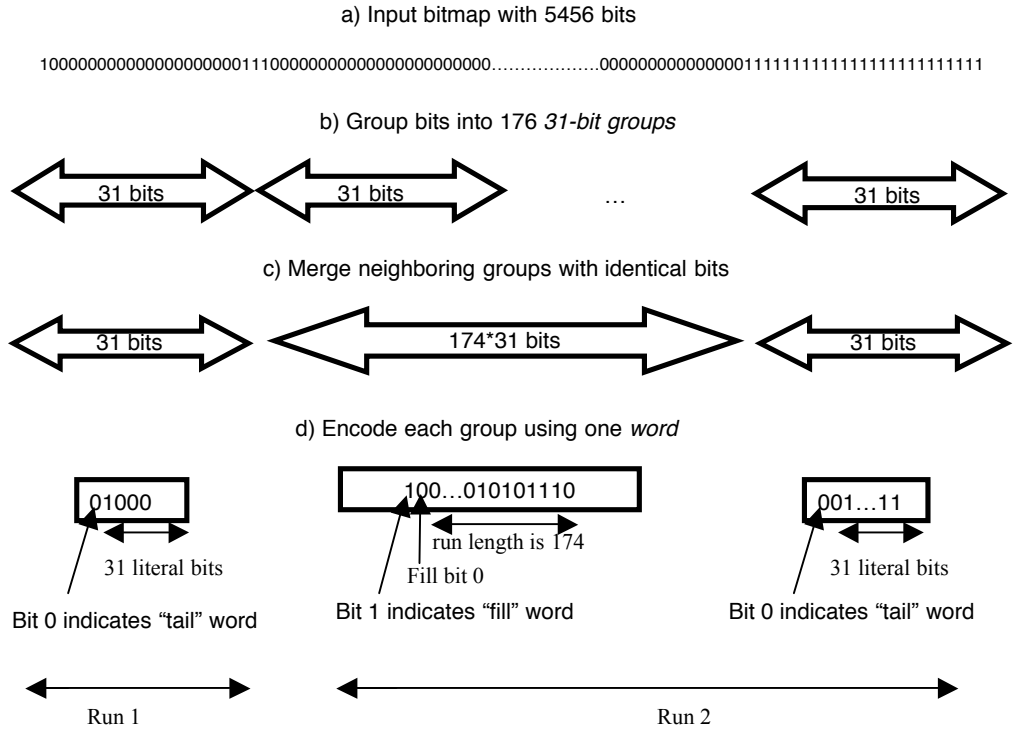

Figure 5: An example WAH encoding of a sequence of 5456 bits on a 32-bit machine.

In theoretical analysis, the query response time on one-dimensional range queries using WAH compressed indices was shown to *grow linearly in the number of hits*. This *time complexity is optimal* for any searching algorithm since one has to return at least the hits, which takes _(*h*) time (where *h* is the number of hits). A variety of well-known indexing methods such as B^+-*trees* and *B*-trees* have the same optimal scaling property. However, compressed bitmap indices have the unique advantage that they can be easily combined to answer *multi-dimensional ad-hoc range queries*, while B^+-trees or B*-trees cannot be combined nearly as efficiently.

In general, the query response time can be broken into I/O time and CPU time. Since WAH compressed bitmaps are larger in size than BBC compressed bitmaps, we would expect that WAH require more I/O time to read compressed bitmaps. For many database operations, the CPU time is negligible compared with the I/O time. It turns out that this is not the case when answering queries with compressed bitmap indices. In a performance experiment Stockinger et al. (2002) compared WAH compressed indices with two independent implementations of BBC compressed indices, one based on Johnson's (1999) code and the other by Wu et al. (2002). The results showed that the total query response time was smaller with WAH compressed bitmap indices than with BBC compressed bitmaps, even on a relatively slow disk system that can only sustain 5 MB/s for reading files from disk. On faster disk systems, the performance advantage of WAH compressed bitmap indices is even more pronounced. Using WAH could be 10 times faster than using BBC.

BITMAP INDEX TUNING

Unless one uses *binary encoding*, it is important to compress the bitmap indices. To build an efficient compressed bitmap index, the three main parameters to consider are: (1) encoding, (2) number of bins, and (3) binning strategy. In the following we present a rule-of-thumb for choosing these three parameters.
The optimal *bitmap encoding technique* depends on the kind of queries that are evaluated. Chan & Ioannidis (1999) showed that *range encoding* is the best bitmap encoding technique for *range-queries*. However, *range encoding* might not always be practical for high-cardinality attributes or for a large number of bins. As we will show in the next section, range-encoded bitmap indices do not compress as well as equality-encoded bitmap indices.

The general rule for choosing the *number of bins* is as follows: *The more bins, the less work during the candidate check*. The reason is fairly straightforward. In general, as the number of bins increases, the number of candidates per bin decreases. Let us consider the following example. Assume the base data follows a uniform random distribution. With a typical page size of 8KB, using the projection index, a page could hold 2048 4-byte words. If one in 1000 words is accessed during the candidate check, it is likely that every page containing the attribute values would be touched (Stockinger et al., 2004). We, thus, suggest using 1000 bins or more.

For *equality encoding* there is an additional trade-off, namely using more bins may also increase the cost of the *index scan*. For *range encoding* the cost of the index scan is not significantly affected by the number of bins because one needs to access no more than two bitmaps to evaluate a range query (Chan & Ioannidis, 1999). Without compression, one would clearly favor *range encoding*. However, with compression, the relative strength is not as obvious. With a WAH compressed equality-encoded index, it was shown that the cost of the index scan is proportional to the number of hits, independent of the number of bitmaps involved (Wu et al., 2006). Because the equality-encoded indices are much easier to compress, this could make the WAH compressed equality-encoded index a preferred choice.

Finally, the *binning strategy* has an impact on the candidate check. The simplest kind of binning, called ***equi-width* binning**, partitions the domain of the indexed attribute into bins of equal size. As a result, each bin might have a different number of entries. ***Equi-depth* binning**, on the other hand, distributes the number of entries equally among the bins. This technique has a better worst-case behavior than equi-width binning but is more costly to build because one

typically has to scan the data first to generate the exact histogram before starting with the binning.

One approach to reduce the cost of building a set of equi-depth bins is to use a *sampled histogram* instead of the *exact histogram*. Another approach is to first build an *equi-width binned index* with more bins than desired, and then *combine the neighboring bins* to form approximate equi-depth bins. However, the second approach might not produce *well-balanced* bins. For example, the attribute *mass fraction* from a supernova simulation is expected to be in the range of 0 and 1. If, for some reason, the mass fraction is not known, scientists typically enter the value -999 to represent a *bad* or *missing value*. In this example, equi-width binning would produce bins starting from -999. This results in too many empty bins and thus cannot be combined to produce *well-balanced* equi-depth bins. In contrast, the approach of sampled histograms is generally more reliable in detecting this type of unusual outliers and typically produces well-balanced bins.

SPACE COMPLEXITY – SIZES OF COMPRESSED BITMAP INDICES

The space complexity of *uncompressed* bitmap indices was studied in (Chan & Ioannidis, 1998 and 1999). In this section, we analyze the size of *compressed* bitmap indices. Our discussion mainly focuses on the WAH compression method since BBC compression was extensively studied in (Johnson, 1999). We give an upper bound of the worst-case size and provide an experimental study of compressed bitmap indices for various application data sets.

Index Size: Worst-Case Behavior

In the previous section we defined a WAH *run* to be a *fill* followed by a *tail*. To make the discussion more concrete, let us assume that a machine word is 32 bits. In this case, a WAH *tail* contains exactly 31 bits from the input bitmap and a WAH *fill* contains a multiple of 31 bits that are all the same (either 0 or 1). Because the bitmap index is known to be efficient for low cardinality attributes, we further restrict our discussion to high cardinality attributes only, say, $c > 100$. In an *equality-encoded* bitmap index, there are c keys (distinct values of the attribute) and thus c bitmaps. We do not know exactly how many bits are set to 1 in each individual bitmap. However, we know that the total number of bits that are 1 is exactly N (the number of rows in the data set). In the worst case, there are $(N+c)$ WAH *runs* in the bitmaps, where N refers to maximum number of tail words (each containing a single bit set to 1) and c refers to the maximum number of runs at the end of each bitmap that are not terminated with a tail word. Each WAH run is encoded by two machine words. Therefore, we need a total of $2(N+c)$ words to represent the bitmaps. Assuming each key is encoded by one word along with one additional word to associate the key with the bitmap, the *total index size* is $2N+4c$ words. In most cases, the attribute cardinality c is much smaller than N. In these cases, the WAH compressed equality-encoded bitmap index size is at worst $2N$ words. With binning, one may use many thousands of bins and the maximum index size would still be no more than $2N$. Since a number of commercial implementations of B-trees are observed to use $3N$ to $4N$ words, the maximum size of compressed bitmap indices is relatively modest. As we will show for real application data, the WAH compressed index is often much smaller than the predicted worst-case sizes.

For WAH compression, in the worst case, about 90% of the bitmaps in a *range-encoded* bitmap index will not be compressible (Wu et al., 2006). Unless one can tolerate very large indices or one knows beforehand that compression would be effective, we generally recommend using *no*

more than 100 bins for range-encode bitmap indices. This guarantees that the size of the bitmap index is at worst the size of a B-tree.

Index Size for Real Application Data Sets

We will now analyze experimentally the size of compressed bitmap indices for various application data sets.

Combustion Data Set

The combustion data set is from a simulation of the auto-ignition of turbulent Hydrogen-air mixture from the TeraScale High-Fidelity Simulation of Turbulent Combustion with Detailed Chemistry (Tera Scale Combustion, 2005). The data set consists of 24 million records with 16 attributes each. For this data set we built equality-encoded and range-encoded bitmap indices with various numbers of equi-depth bins. Figure 6 shows the average size of the compressed bitmap indices per attribute. We can see that equality-encoded bitmap indices with 1000 bins and the range-encoded bitmap indices with 100 bins have about the same size as the base data. Note that the size of an uncompressed bitmap index with 100 bins is about 3 times as large as the base data. With 1000 bins, the size of the uncompressed bitmap index is about 30 times larger. This shows that the WAH compression algorithm works well on this data set.

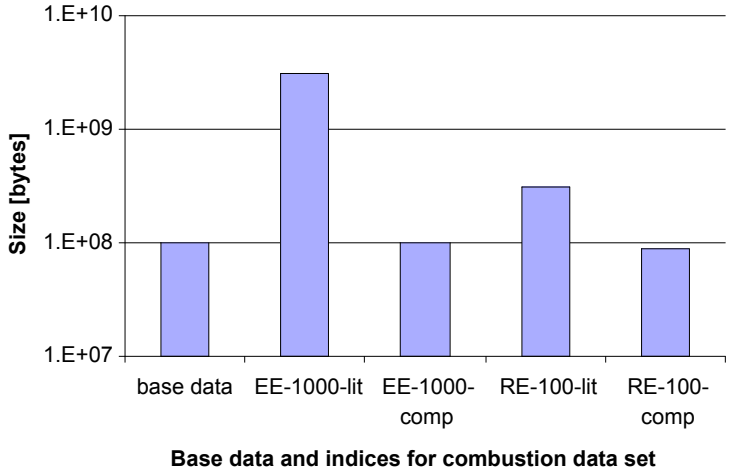

Figure 6: Size of base data compared with bitmap indices. EE = equality encoding; RE = range encoding; lit = literal (no compression); comp = with compression.

High-Energy Physics Data Set

Our second data set is from a high-energy physics experiment at the Stanford Linear Accelerator Center. It consists of 7.6 million records with 10 attributes. Figure 7 shows the size of the compressed bitmap indices. We notice that the size of the range-encoded bitmap index with 100 bins is about twice as large as the base data. The equality-encoded bitmap index with 1000 bins is about 30% smaller than the base data.

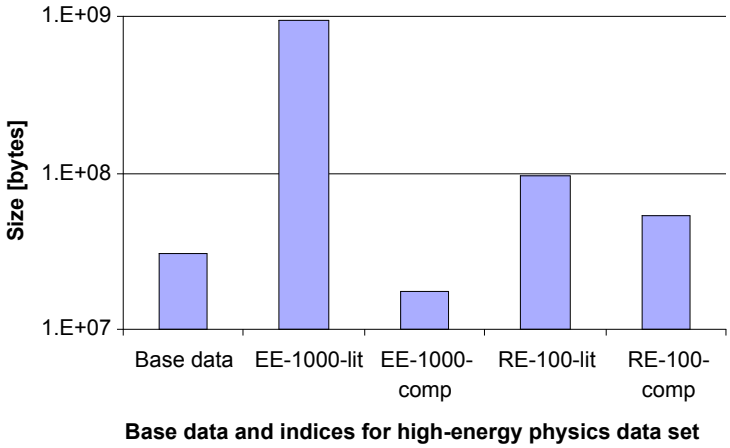

Figure 7: Size of base data compared with bitmap indices. For an explanation of the legend see Figure 6.

Typically, the records from these high-energy physics experiments are not correlated with each other. Thus, it is generally hard for the **run-length encoding** to be effective. This is why the index sizes for *range encoding* are relatively large compared with the previous data sets. However, *equality encoding* compresses very well for this physics data set.

Overall, we see that the actual bitmap index sizes are considerably smaller than the base data sizes and less than the sizes of typical commercial implementations of B-trees (that are often three to four times the size of the base data).

TIME COMPLEXITY - QUERY RESPONSE TIME

In this section we are focusing on the two basic encoding methods, namely *equality encoding* and *range encoding*. We have chosen these two encoding methods for the following reason. *Equality encoding* showed to be the most space efficient method. *Range encoding*, on the other hand, is the most time efficient method for one-sided range queries (Chan & Ioannidis, 1998) that we use in our experiments.

Analyses have shown that the worst case query response time to answer a one-dimensional range query using a WAH compressed basic bitmap index (equality-encoded without binning) is a linear function of the number hits (Wu et al., 2006). The analyses also indicate that the worst-case behavior is for attributes following a uniform random distribution. Figure 8 plots the query response time against the number of hits for a set of queries on two attributes with different attribute cardinalities. The data values for the two attributes are randomly distributed in the range of [0;100] and [0; 10,000] respectively. We see that in both cases the timing measurements follow straight lines, which is theoretically *optimal*.

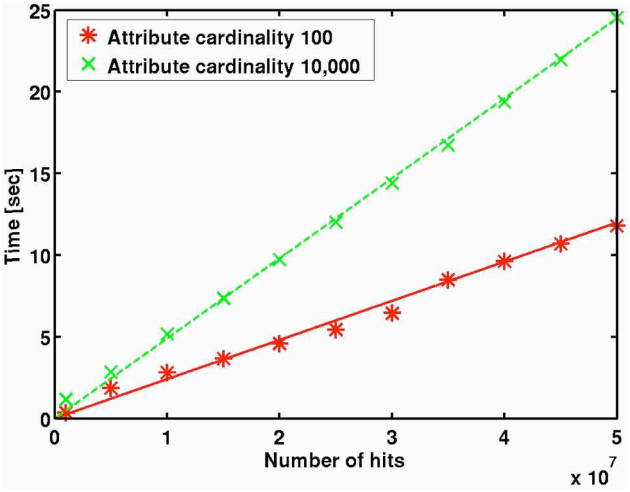

Figure 8: Time (in seconds) to answer a one-dimensional range query using a WAH compressed bitmap index is a linear function of the number of hits.

In the remainder of this section we present more timing measurements to compare the query response time of equality-encoded and range-encoded bitmap indices. All indices are compressed with WAH compression. Since the results for the two data sets are similar, we only report on the measurements based on the larger and thus more challenging combustion data set. We use the *projection index* as the base line for all the comparisons. We note that this is a good base line since the projection index is known for outperforming many multi-dimensional indices.

In the next set of experiments we measure the *query size* with the *query box size*. A query box is a *hypercube formed by the boundaries* of the range conditions in the attribute domains. We measure the query box size as the fraction of the query box volume to the total volume of the attribute domains. If all attributes have uniform distribution, then a query box size of 0.01 indicates that the query would select 1% of the data set. We say a query is more selective if the query box size is smaller.

Figure 9 shows the response time (in seconds) for 2- and 10-dimensional queries with various query box sizes. For all experiments the query box size was chosen randomly and covers the whole domain range. In general, we see that the query processing time for the bitmap indices decreases as the queries become more selective. On the other hand, the query processing time for the projection index stays constant as the selectivity changes. For all query dimensions the range-encoded bitmap index with 100 bins shows the best performance characteristics, however, sometimes at the cost of a larger index. In case the storage space is a limiting factor, it is better to choose equality-encoded bitmap indices with 1000 bins (see Figure 7). As we can see in Figure 9, the performance of equality-encoded bitmap indices is not significantly different from the performance of range-encoded bitmap indices.

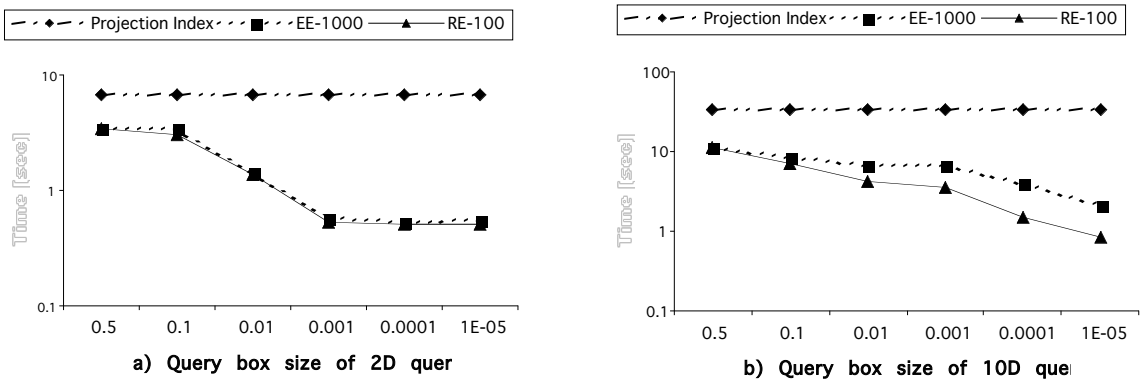

Figure 9: Multi-dimensional queries with various bitmap indices. EE-1000: equality encoding with 1000 bins, RE-100: range encoding with 100 bins.

KEY FEATURES IN COMMERCIAL PRODUCTS

Due to the considerable amount of work involved in producing and maintaining a robust commercial software system, only the most efficient and proven indexing technologies make their way into a commercial DBMS. In this section, we give a short review of the key bitmap indexing technologies currently used by various well-known commercial products. This is not meant to be an exhaustive survey. Our main interest is to see what kind of bitmap indexing technology is missing and which technology may likely make an impact on commercial products.

The first commercial product to use the name *bitmap index* is *Model 204*. O'Neil has published a description of the indexing method in (O'Neil, 1987). Model 204 implements the basic bitmap index. It has no binning or compression. Currently, Model 204 is marketed by Computer Corporation of America. ORACLE has a version of compressed bitmap indices in its flagship product since version 7.3. They implemented a proprietary compression method. Based on the observed performance characteristics, it appears to use *equality encoding* without binning.

Sybase IQ implements the bit-sliced index (O'Neil & Quass, 1997). Using the terminology defined in Sections 2 and 3, Sybase IQ supports unbinned, binary encoded, uncompressed bitmap indices. In addition, it also has the basic bitmap index for low-cardinality attributes. IBM DB2 implements a variation of the binary encoded bitmap index called *Encode Vector Index*. IBM Informix products also contain some versions of bitmap indices for queries involving one or more tables. These indices are specifically designed to speed up join-operations and are

commonly referred to as *join indices* (O'Neil and Quass, 1997). InterSystems Corp's Cache also has bitmap index support since version 5.0.

Even though we do not have technical details on most of these commercial products, it is generally clear that they tend to use either the basic bitmap index or the bit-sliced index. Strategies like *binning* and *multi-component encoding* are not used partly because there is no robust strategy to select parameters like the number of bins or the number of components that suits different applications.

SUMMARY AND OPEN PROBLEMS

In this chapter, we reviewed a number of recent developments in the area of bitmap indexing technology. We organized much of the research work under the three orthogonal categories of encoding, compression and binning. We also provided a brief overview of commercial bitmap index implementations by major vendors.

Most of the indexing methods reviewed were designed to efficiently answer multi-dimensional range queries. However, they are also efficient for other types of queries, such as joins on foreign keys and computations of aggregates (O'Neil & Quass, 1997).

Despite the success of bitmap indices, there are a number of important questions that remain to be addressed. For example, is there an efficient bitmap index for similarity queries? How to automatically select the best combination of encoding, compression and binning techniques? How to use bitmap indices to answer more general join queries?

Research work on bitmap indices so far has concentrated on answering queries efficiently, but has often neglected the issue of *updating* the indices. Clearly, there is a need to update the indices as new records are added. Efficient solutions to this issue could be the key to gain a wider adaptation of bitmap indices in commercial applications.

REFERENCES

- Antoshenkov, G. (1994). Byte-aligned Bitmap Compression. *Technical Report*, Oracle Corp., 1994. U.S. Patent number 5,363,098.
- Antoshenkov, G. & Ziauddin, M. (1996). Query Processing and Optimization in ORACLE RDB, *VLDB Journal*, 5, 229-237.
- Berchtold, S., & Boehm, C., & Kriegl, H.-P. (1998) The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. *SIGMOD Record*, 27(2), 142-153.
- Chan, C.-Y., & Ioannidis, Y.E. (1998). Bitmap Index Design and Evaluation. *SIGMOD*, Seattle, Washington, USA, ACM Press.
- Chan, C.-Y., & Ioannidis, Y.E. (1999). An Efficient Bitmap Encoding Scheme for Selection Queries, *SIGMOD Conference*, Philadelphia, Pennsylvania, USA, ACM Press.
- Chaudhuri, S., & Dayal, U. (1997). An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), 65-74.
- Chaudhuri, S. & Dayal, U., & Ganti, V. (2001). Database Technology for Decision Support Systems. *Computer*, 34(12), 48-55.
- Comer, D. (1979). The ubiquitous B-Tree. *Computing Surveys*, 11(2), 121-137.
- FastBit (2005), Retrieved January 11, 2006, from http://sdm.lbl.gov/fastbit.
- Gaede, V & Guenther, O. (1998) Multidimensional Access Methods. ACM *Computing Surveys*, 30(2), 170—231.

- Johnson, T. (1999). Performance Measurements of Compressed Bitmap Indices. *International Conference on Very Large Data Bases (VLDB)*, Edinburgh, Scotland. Morgan Kaufmann.
- Keim, D., & Hinneburg, A. (1999). Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. *International Conference on Very Large Data Bases (VLDB)*, San Francisco. Morgan Kaufmann.
- Kiyoki, Y., & Tanaka, K., & Aiso, H., & Kamibayashi, N. (1981). Design and Evaluation of a Relational Data Base Machine Employing Advanced Data Structures and Algorithms. *Symposium on Computer Architecture*, Los Alamitos, CA, USA. IEEE Computer Society Press.
- Knuth, D. E. (1998). The Art of Computer Programming, Volume 3, Addison Wesley.
- Koudas, N. (2000). Space Efficient Bitmap Indexing. *Conference on Information and Knowledge Management* (*CIKM),* McLean, V, USA. ACM Press.
- O'Neil, P. (1987). Model 204 Architecture and Performance. Workshop in High Performance Transaction Systems, Asilomar, California, USA. Springer-Verlag.
- O'Neil, P., & Quass, D. (1997). Improved Query Performance with Variant Indexes. *International Conference on Management of Data (SIGMOD 1997)*, Tucson, Arizona, USA. ACM Press.
- O'Neil, P. (1997). Informix Indexing Support for Data Warehouses. *Database Programming and Design*, 10(2), 38-43.
- Rotem, D. & Stockinger, K. & Wu, K. (2005a) Optimizing I/O Costs of Multi-Dimensional Queries using Bitmap Indices, *International Conference on Database and Expert Systems Applications (DEXA)*, Copenhagen, Denmark, Springer Verlag.
- Rotem, D. & Stockinger, K. & Wu, K. (2005b) Optimizing Candidate Check Costs for Bitmap Indices**,** *Conference on Information and Knowledge Management (CIKM)*, Bremen, Germany, November 2005, ACM Press.
- Stockinger, K., & Wu, K., & Shoshani, A. (2002). Strategies for Processing ad hoc Queries on Large Data Sets. International Workshop on Data Warehousing and OLAP (*DOLAP),* McLean, Virginia, USA.
- Stockinger, K., & Wu, K., & Shoshani, A. (2004). Evaluation Strategies for Bitmap Indices with Binning. *International Conference on Database and Expert Systems Applications (DEXA),* Zaragoza, Spain. Springer-Verlag.
- Stockinger, K., & Shalf, J., & Bethel, W., & Wu, K. (2005) DEX: Increasing the Capability of Scientific Data Analysis Pipelines by Using Efficient Bitmap Indices to Accelerate Scientific Visualization, *International Conference on Scientific and Statistical Database Management* (SSDBM), Santa Barbara, California, USA, June 2005, IEEE Computer Society Press.
- TeraScaleCombustion (2005). TeraScale High-Fidelity Simulation of Turbulent Combustion with Detailed Chemistry. Retrieved January 11, 2006 from http://scidac.psc.edu.
- Wong, H.K.T., & Liu, H. -F., & Olken, F., & Rotem, D., & Wong, L. (1985). Bit Transposed Files. *International Conference on Very Large Databases* (*VLDB)*, Stockholm, Sweden. Morgan Kaufmann.
- Wu, K., & Otoo, E.J., & Shoshani, A. (2002). Compressing Bitmap Indexes for Faster Search Operations. *International Conference on Scientific and Statistical Database Management (SSDBM),* Edinburgh, Scotland, UK, IEEE Computer Society Press.

- Wu, K., & Otoo, E.J., & Shoshani, A. (2004). On the Performance of Bitmap Indices for High Cardinality Attributes. *International Conference on Very Large Data Bases (VLDB),* Toronto, Canada. Morgan Kaufmann.
- Wu, K., & Otoo, E., & Shoshani, A. (2006). An Efficient Compression Scheme for Bitmap Indices. *Technical Report LBNL-49626*. To appear in *ACM Transactions on Database Systems (TODS)*.
- Wu, M.-C., & Buchmann, A.P. (1998). Encoded Bitmap Indexing for Data Warehouses. *International Conference on Data Engineering (ICDE)*, Orlando, Florida, USA. IEEE Computer Society Press.