FastBit: Interactively Searching Massive Data

K Wu[1], S Ahern[2], E W Bethel[1,3], J Chen[4], H Childs[5], E Cormier-Michel[1], C Geddes[1], J Gu[1], H Hagen[3,6], B Hamann[1,3], W Koegler[4], J Lauret[7], J Meredith[2], P Messmer[8], E Otoo[1], V Perevoztchikov[7], A Poskanzer[1], Prabhat[1], O Rübel[1,3,6], A Shoshani[1], A Sim[1], K Stockinger[1,10], G Weber[1] and W-M Zhang[9]

[1] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.
[2] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA.
[3] University of California - Davis, Davis, CA 95616, USA.
[4] Sandia National Laboratories, Livermore, CA 94551, USA.
[5] Lawrence Livermore National Laboratory, Livermore, CA 94550, USA.
[6] Technische Universität Kaiserslautern, Erwin-Schrödinger-Straße, Kaiserslautern, Germany.
[7] Brookhaven National Laboratory, Upton, NY 11973, USA.
[8] Tech-X Corporation, Boulder, CO 80303, USA.
[9] Kent State University, Kent, OH 44240, USA.
[10] Current address: Credit Suisse, Zurich, Switzerland.

Abstract. As scientific instruments and computer simulations produce more and more data, the task of locating the essential information to gain insight becomes increasingly difficult. FastBit is an efficient software tool to address this challenge. In this article, we present a summary of the key techniques, namely bitmap compression, encoding and binning. The advances in these techniques have led to a search tool that can answer structured (SQL) queries orders of magnitude faster than popular database systems. To illustrate how FastBit is used in applications, we present three examples involving a high-energy physics experiment, a combustion simulation, and an accelerator simulation. In each case, FastBit significantly reduces the response time and enables interactive exploration on terabytes of data.

1. Introduction

Current generations of scientific instruments [1, 2] and computer simulations [3, 4] are capable of producing massive amounts of data. However, a relatively small percentage of records often hold the key to the insight sought by scientists. For example, the high-energy physics experiment STAR [5] collects billions of events of colliding particles, however, only a few hundred of them might be the rare events having the unique signature of Quark-Gluon Plasma – finding evidence of Quark-Gluon Plasma is one of the key scientific objectives of STAR. Locating a relatively small number of data records distributed throughout a huge collection is a common challenge in many scientific fields. Database Management Systems (DBMS) are designed to handle such tasks, however, existing DBMS are not well-suited for scientific data explorations.

The existing DBMS systems are designed for transactional applications, where the data records are frequently modified such as modifying one's bank account during an ATM withdrawal. In contrast, scientific applications typically do not perform such transactions. Most interactions with a scientific data set are to select a subset of records for further analyses, for example, selecting a number of collision events with certain properties and computing

RID	**A**	bitmap index			
		=0	=1	=2	=3
1	0	1	0	0	0
2	1	0	1	0	0
3	2	0	0	1	0
4	2	0	0	1	0
5	3	0	0	0	1
6	3	0	0	0	1
7	1	0	1	0	0
8	3	0	0	0	1
		b_1	b_2	b_3	b_4

Figure 1. An illustration of the basic bitmap index for a column **A** that can only take on four distinct values from 0 to 3. Note RID is the short-hand for "row identifiers".

a histogram of their energy distribution, or selecting regions with a high temperature in a combustion simulation and computing the amount of heat generated from each region. Such analysis tasks are generally supported by a data warehousing system and require a different data access technology [6–8].

A familiar search technology that routinely handles a large amount of data is the web search engine [9, 10]. The key underlying technology is the inverted index designed for text data. Since the bulk of scientific data are numerical values, this indexing technology is not applicable. However, other technologies employed by search engines such as the parallel iterators can be used effectively for scientific applications.

We have been working on a set of techniques specifically designed for scientific data. They are based on the bitmap index designed to answer queries efficiently on read-only data [11, 12]. One significant shortcoming of the earlier bitmap indexes is that they are only effective on low-cardinality variables that has a small number of distinct values, such as the gender of a customer or the state of a customer's address. However, scientific data usually contains variables with very high cardinalities, such as floating-point values that hardly ever repeat. In this paper, we review key techniques to make bitmap indexes efficient on high-cardinality variables.

We have implemented the new techniques in a package called FastBit[1]. Since the release of the software in late 2007, it has been download thousands of times, received an R&D 100 award, contributed to a number of PhD theses [13, 14], and produced a number of academic publications. It has also been integrated with a number of popular data management and analysis systems, such as HDF5 [15, 16] and VisIt [17, 18]. In Section 3, we review three applications involving the authors. Note that there are a number of independent publications [19, 20]. For example, a new indexing method for geographical data warehouses based on FastBit performed 10 - 20 times faster than existing techniques [20]. In a virtual screening software, FastBit helped to speed up the application 100s of times [19].

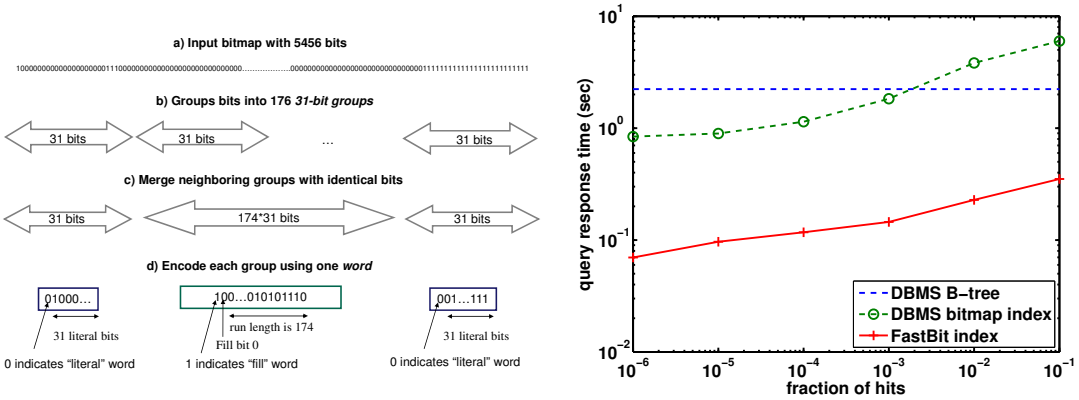
2. Core Bitmap Indexing Techniques

In this section we review the core bitmap indexing techniques. We start with a brief description of the basic bitmap index.

2.1. Basic bitmap index

Figure 1 shows a logical view of the basic bitmap index. Conceptually, this index contains the same information as a B-tree [12, 21]. The key difference is that a B-tree would store a list of

[1] FastBit software is available from `https://codeforge.lbl.gov/projects/fastbit/`.



(a) an illustration of WAH (b) time (s) to answer queries

Figure 2. The effects of compression on query response time. The faster WAH compression used in FastBit reduces the query response time by an order of magnitude [26, 27].

Row IDentifiers (RIDs) for each distinct value of column **A**, whereas a bitmap index represents the same information as sequences of bits, which we call *bitmaps*. In this basic bitmap index, each bitmap corresponds to a particular value of the column. A bit with value 1 indicates that a particular row has the value represented by the bitmap.

The bitmap indexes are particular useful for query-intensive applications, such as data warehousing and OLAP [6]. One of the key reasons is that queries can be answered with bitwise logical operations on the bitmaps. In the example shown in Figure 1, a query "$\mathbf{A} < 2$" can be answered by performing bitwise OR on b_1 and b_2 (i.e., $b_1 \mid b_2$). Since most computer hardware support such bitwise logical operations efficiently, the queries can be answered quickly in general. Another key reason is that answers from different bitmap indexes can be easily combined. This is because the answer from each bitmap index is a bitmap and combining the different answers simply requires additional bitwise logical operations. Since combining answers from different indexes swiftly is such an important consideration, a number of DBMS that do not support bitmap indexes, such as PostgreSQL and MS SQL Server, even convert intermediate solutions to bitmaps to combine them more effectively.

The biggest weakness of the basic bitmap index is that its size grows linearly with the number of distinct values in the column being indexed. Next we review three sets of strategies to control the index sizes and improve the query response time, namely, compression, encoding and binning.

2.2. Bitmap Compression

Each bitmap in a bitmap index usually contains a large number of 0s, making it easy to compress. To answer queries efficiently, researchers have developed specialized compression methods that offer faster bitwise logical operations and consequently faster query response time [22, 23]. The most widely used bitmap compression method is Byte-aligned Bitmap Code (BBC) [24]. We developed a new bitmap compression method called Word-Aligned Hybrid (WAH) code and showed it to perform bitwise logical operations more than 10 times faster than BBC [25, 26].

WAH gains its speed partly from its simplicity. For long sequences of 0s or 1s, it uses run-length encoding to represent them and for relatively short sequences of mixed 0s and 1s, it represents the bits literally. Hence, it is a hybrid of two methods. Another key feature that enables it to achieve performance is that the compressed data is word-aligned. More specifically, WAH compressed data contains two types of words; *literal* words and *fill* words.

As illustrated in Figure 2(a), a *literal* word contains one bit to indicate its type and uses the remaining bits to store the bitmap literally. A *fill* word similarly needs 1 bit to indicate its type. It uses another bit to indicate whether the bits are all 0s or all 1s, and the remaining bits are used to store the number of bits in a bitmap it represents. The number of bits represented by a WAH *fill* word is always a multiple of the number of bits stored in a literal word.

Figure 2(b) illustrates the effects of compression on overall query response time. In this test, the commercial implementation of the bitmap index (marked as "DBMS bitmap index") uses BBC compression, while "FastBit index" uses WAH compression. The query response time reported are average time values over thousands of ad hoc range queries that produce the same number of hits. Overall, the WAH compressed indexes answer queries about 14 times faster than the commercial bitmap indexes.

2.3. Bitmap Encoding

Bitmap encoding techniques manipulate the bitmaps produced by the basic bitmap index to reduce either the number of bitmaps in an index or the number of bitmaps needed to answer a query. For example, to answer a one-sided range query of the form "$\mathbf{A} < 3$" on the data shown in Figure 1, one needs to OR three bitmaps b_1, b_2 and b_3. It is possible to store C bitmaps that correspond to $\mathbf{A} \leq a_i$ for each of the C distinct values of $\mathbf{A}$, where C is called the *column cardinality* of $\mathbf{A}$. Such a bitmap index would have the same number of bitmaps as the basic bitmap index, but can answer each one-sided range query by reading just one bitmap. This is the *range encoding* [11]. Along with the *interval encoding* and the *equality encoding* (used in the basic bitmap index), there are three basic bitmap encoding methods. Next, we briefly explore strategies to compose more advanced encodings using these basic methods.

The two common strategies of composing bitmap encodings are multi-component encoding [11] and multi-level encoding [28, 29]. The central idea of multi-component encoding is to break the key value corresponding to a bitmap into multiple components in the same way an integer number is broken into multiple digits in a decimal or binary representation. In general, using more components reduces the number of bitmaps needed, which may reduce the index size. Carrying this to the extreme, we can make all base sizes as small as possible (where the minimum is 2). This particular multi-component encoding can be optimized to be the binary encoding [30, 31]. This encoding produces the minimum number of bitmaps and the corresponding index size is the smallest without compression.

The new class of encoding method we proposed was the multi-level index composed of a hierarchy of nested bins on the base data. Each level in such an index is a complete index on its own, and can answer queries on its own or in combination with other levels. This flexibility can be exploited to achieve better performance [32]. Figure 3 shows some timing measurements of five multi-level and multi-component indexes. These timing measurements indicate that two-level encodings, especially the range-equality encoding RE and the interval-equality encoding IE, can be as much as ten times faster than the basic bitmap index (marked E1). On average, the two-level encoded indexes are about 3 to 5 times faster than the basic bitmap index and the binary encoded index (BN).

2.4. Binning

Scientific data often computed or collected with high precision. For example, the temperature and pressure in a combustion simulation are computed with 64-bit floating-point values having 16 decimal digits of precision. To capture the full precision of the data in a bitmap index, one typically needs millions of bitmaps in the basic bitmap index. Such indexes can be large and slow to work with, even with the best compression and bitmap encoding techniques. We observe that an exploratory process does not usually query data at the full precision. For example, a typical query involving pressure is of the form "pressure $> 2 \times 10^7$ Pascal." In this case,

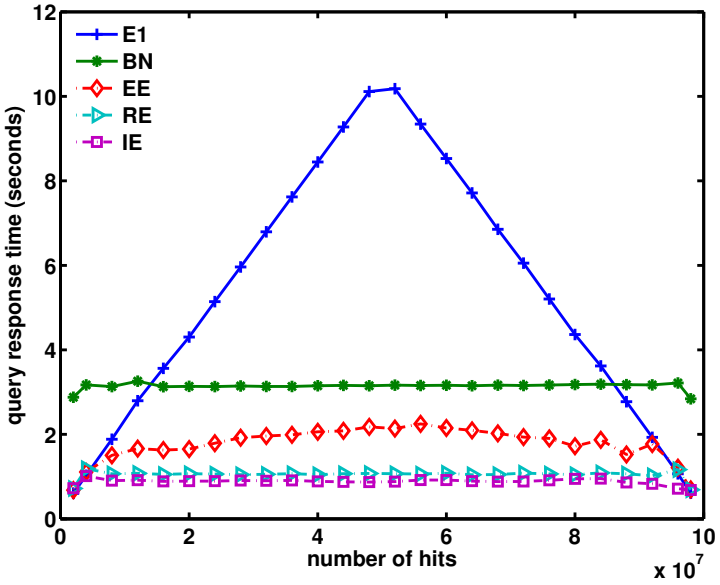
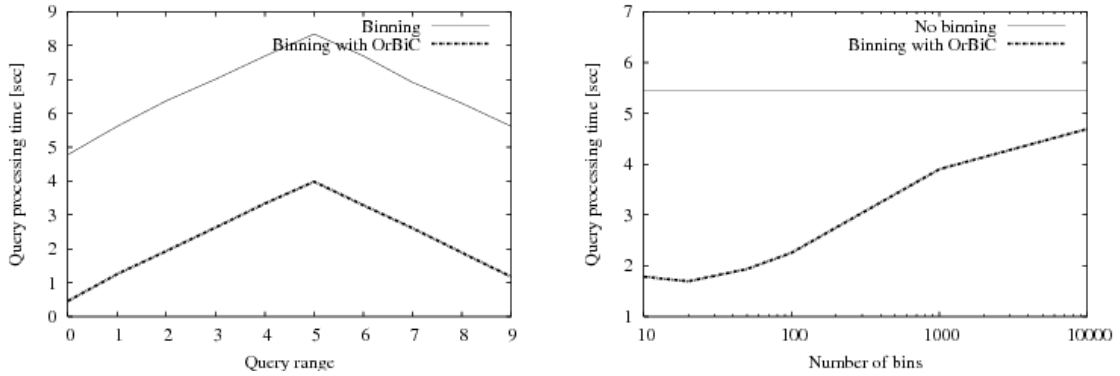


Figure 3. The average query response time of five different bitmap encoding methods with WAH compression (BN: binary encoding, E1: the basic one-component equality encoding, EE: two-level equality-equality encoding, RE: two-level range-equality encoding, IE: two-level interval-equality encoding) [32].

the constant in the query expression has only one significant digit. To answer all such queries with one-significant-digit constants, we only need to make sure all the values are binned to one significant digit, which only requires 9 bins for each distinct exponent in the data. Even if the user requires more than one significant digit, typically, 2 or 3 significant digits are enough. The number of bins needed are still modest.

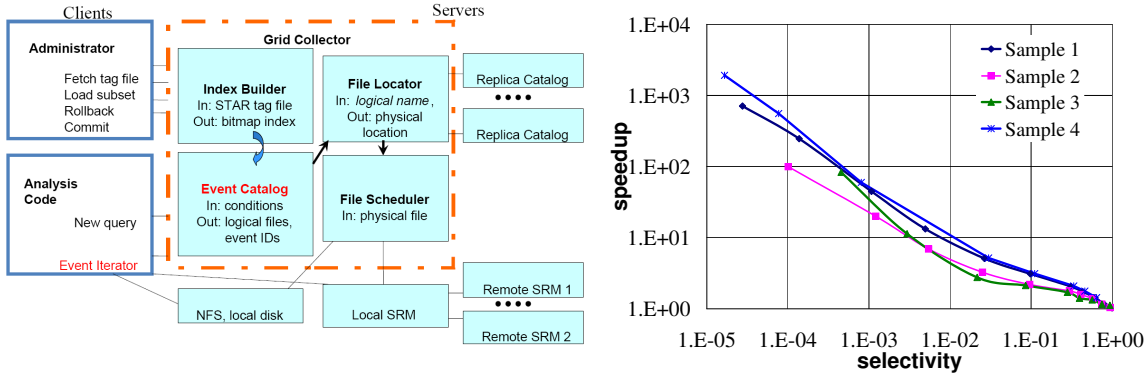
Binning can reduce the number of bitmaps in an index and improve the query response time as well. However, for some queries, we have to go back to the base data in order to answer them accurately. For example, if we have 99 equal-width bins for pressure between 0 and 10^9 (with 100 bin boundaries at $i \times 10^7, i = 0, \ldots, 100$), then the query condition "pressure $> 2.5 \times 10^7$" cannot be resolved with the index only. More specifically, bins 0 and 1 contain records that satisfy the query condition, and bins 3 and onwards contain records that do not satisfy the condition, but records in bin 2 (representing $2 \times 10^7 \leq$ pressure $< 3 \times 10^7$) need to be examined further. In this case, we say the records in bin 2 are *candidates* of the query. The process of determining which candidate actually satisfies the condition is called a *candidate check*. When a candidate check is needed, it often dominates the total query response time.

We have studied different approaches to minimize the impact of candidate checks. One is to reorder the expression being evaluated to minimize the overall cost of candidate checks for a query [34]. Another approach is to place the bin boundaries to minimize the cost of evaluating a fixed set of queries [35–38]. More recently, we proposed a new approach to more directly reduce the candidate check cost by providing a clustered copy of the base data named Order-preserving Bin-based Clustering (OrBiC) [33]. This approach organizes all values of each bin contiguously on disk to reduce the I/O time needed for candidate checks. Figure 4 shows some performance numbers to illustrate the advantages of this new approach. Figure 4(a) shows the binned index with OrBiC outperforms the one without OrBiC for all query condition tested, and Figure 4(b) shows the average time used by a binned index with OrBiC to be less than that used by the precise index (with no binning).



(a) individual query (b) average time

Figure 4. Time needed to process range queries using indexes with and without Order-preserving Bin-based Clustering (OrBiC) [33].



(a) Schematics of Grid Collector (b) Speedup using Grid Collector

Figure 5. Using Grid Collector can significantly speed up analysis of STAR data [39, 40].

3. Extended Technologies to Support Applications

After reviewing the core techniques behind the FastBit searching capability, we now show how this basic searching capability enables more complex searching operations.

3.1. Smart Parallel Iterator

A large data set is often organized as many files each containing a number of data records, and the files are usually stored on massive storage systems. To perform an analysis, one needs a cluster of computers to hold the necessary files. The analysis system needs to retrieve and replicate the files on the cluster. Managing the file replication, and extracting the desired records can be challenging. The mapReduce systems from google and Yahoo! automate some of these operations [41]. The core concept in such a system is the parallel iterator that coordinates the accesses to files and records. In this example, we demonstrate how FastBit fits into an efficient parallel iterator system.

The version of parallel iterator system we built was called Grid Collector. The prototype was used in the STAR experiment and integrated into the STAR Data Analysis Framework [5, 40].

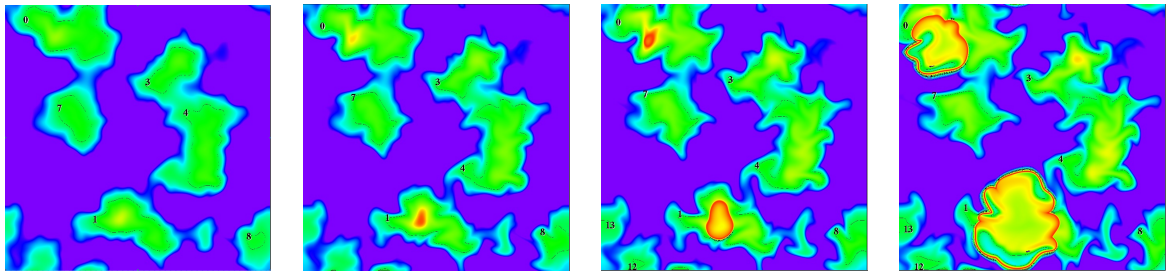


Figure 6. Tracking the evolution of ignition kernels in a combustion simulation [43].

The parallel iterator extended the functionality of the Event Iterator in the STAR analysis framework, and could be used transparently with existing analysis programs. Figure 5(a) shows the schematics of Grid Collector. FastBit is responsible for building the indexes for the Event Catalog and resolving the selection criteria specified by the users to identify files and events. The Grid Collector server also scheduled the file movements and cached files with the storage resource managers [42].

Figure 5(b) shows the observed speedup values versus the selectivity of the analysis task, where the baseline is the time needed to run the same analysis without Grid Collector. The selectivity is the fraction of events in these files that are selected for the analysis task. The Grid Collector speeds up analysis tasks primarily by reducing the amount of disk pages accessed. When one out of 1000 events is selected, the speedup values are observed to be between 20 and 50. When one in ten events is used in an analysis job, the observed speedup is more than 2-fold.

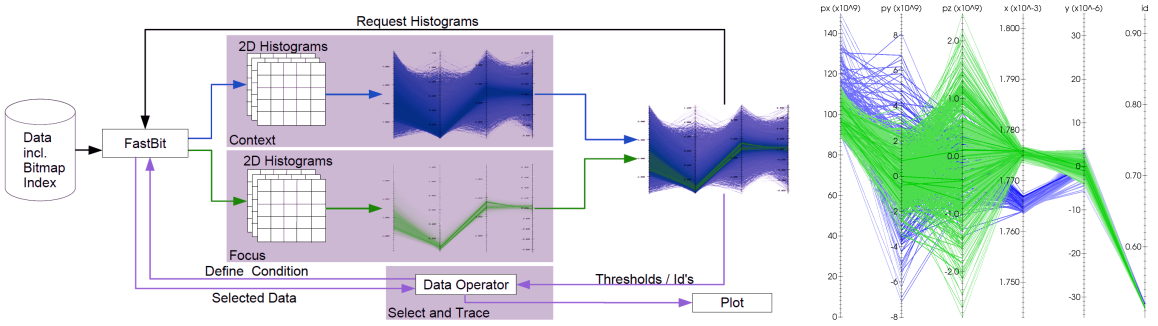
3.2. Tracking spatial features

The previous example contains independent data records, however, many others have correlated records and the search operations depend on this correlation. For example, the records produced by a combustion simulation are related to each other by a mesh, to find and track ignition kernels, we need the connectivity information of this underlying mesh (see Figure 6) [43, 44]. FastBit indexes are uniquely able to take advantage of the underlying correlations among the data records because they preserve the ordering of data provided by the user.

We divide the task of tracking ignition kernels into three steps: 1) finding points (records) satisfying the user provided conditions for ignition kernels, 2) grouping the points into connected components, and 3) tracking the evolution of the components by computing the overlap among them. On a casual examination, FastBit indexes seem to only offer help in the first step. Indeed, many database indexes can only accelerate the first step. However, FastBit indexes and the underlying compressed bitmap data structure can speed up all three steps.

In step 2), we are able to take advantage of the compressed answers generated from step 1) to provide a compact representation of the points in the regions of interest. Instead of recording each data point separately, we store *line segments* consisting of multiple consecutive points [43]. We further provide an efficient union-find data structure to accelerate the operations of step 2) [45]. At the end of step 2), we again represent each connected component as a compressed bitmap, which are used to compute overlaps through bitwise AND operations.

In general, the first two steps can find any regions of interest. We observed that the time needed is a linear function of the number of line segments [43]. Since the number of line segments is bounded by the number of points on the surface of the regions, our algorithm for finding regions of interest is bounded by a linear function of the number of points on the surface, instead of the number of points in the regions. For well-defined regions, where the number of point inside is much larger than the number of points on the surface, our approach has a significant advantage.



(a) system schematics (b) parallel coordinate

Figure 7. The schematics of the parallel coordinate system used for selecting and displaying particles from a Laser Wake-Field Accelerator simulation (a) and a set of parallel coordinates for a time step of the simulation (b) [18].

3.3. Dynamic conditional histograms

Aside from its searching capability, FastBit also has a set of efficient functions for computing conditional histograms [18, 46, 47]. In this example of visual exploration of Laser Wake-Field Accelerator (LWFA) simulation data, we illustrate how this capability is used to produce parallel coordinate plots and facilitate the tracking and understanding of particle bundles in LWFA[2]. The new visual analysis system reduced the turn-around time from hours to seconds.

Figure 7(a) illustrates the use of FastBit to produce a parallel coordinate plot in this application. It used FastBit to select the records according to user specified conditions and to compute the histograms on the selected records. For example, the context in Figure 7(b) (shown in blue) was selected with "$\text{px} > 8.872 \times 10^{10}$." The focus (shown in green) was selected with an additional condition of "$\text{x} > 1.77 \times 10^{-3}$." The parallel coordinate plots contain only particles with a high forward momentum px (shown as the first axis on the left) $> 8.872 \times 10^{10}$. These particles form two well-separated groups on the x-axis (shown as the 4th axis from the left), an important feature for understanding the particle bundles in LWFA.

These parallel coordinate plots are generated using 2-D histograms involving neighboring axes. FastBit produced these histograms with the data records satisfying the user specified conditions and removed the need for user (or the visualization system) to read the whole data set then perform the filtering. This reduced the volume of data accessed and improved the interactivity of the visual exploration [18].

In the same application, FastBit was also used to track particles through time and select data for volume rendering and other computations. For example, after a localized group of accelerated particles was found in one time step, scientists might track its history to understand how it was formed. To accomplish this task, we first extracted the identifiers of the particles, then found particles with these identifiers in other time steps. This search operation involved many thousands of particle identifiers over millions of particle in the base data. The ability to handle this operation efficiently is another feature of FastBit.

4. Summary and Future Plans

FastBit employs a number of advanced bitmap indexing techniques to significantly reduce the query response time on large scientific data. Its advantages have not only been observed in timing measurements, but also proven through theoretical computational complexity analyses [**?**, 23, 32]. Furthermore, we have extended FastBit to include functions for handling simple meshes

[2] Note that the user data is stored in HDF5 files [15, 16] and visualization is performed with VisIt [17].

information and computing conditional histograms. All these are useful features in analyzing massive data sets. In addition, FastBit software is available open source so that anyone can derive from it or extend it for their own work.

A number of new features are currently planned for FastBit, such as handling of queries that join multiple tables. To support more applications, we are also working on integrating information about irregular and semi-regular meshes for queries involving spatial data. We are also studying ways to perform text searches using compressed bitmap indexes.

FastBit shares a number of shortcomings of all other bitmap indexes; the most important of which is that they are slow to update after modifications of a few existing records. This prevents it from being used effectively in applications where data records change frequently. However, if the changes are limited to relatively new records, FastBit could work effectively by avoiding indexing those new records.

Acknowledgments

This work was supported in part by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, the Scientific Discovery through Advanced Computing (SciDAC) program's Scientific Data Management (SDM) center, Visualization and Analytics Center for Enabling Technologies (VACET) and the COMPASS project. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The authors also gratefully acknowledge FastBit users for their efforts to test and debug the software package, and to find new and creative ways of using the software.

References

[1] Anticic T, Barroso V, Carena F, Carena W, Chapeland S, Cobanoglu O, Denes E, Divia R, Fuchs U, Kiss T, Makhlyueva I, Ozok F, Roukoutakis F, Schossmaier K, Soos C, Vyvre P V and Vergara S 2008 Commissioning of the ALICE data acquisition system *CHEP'07* (IOP Publishing)

[2] Gameiro S, Crone G, Ferrari R, Francis D, Gorini B, Gruwe M, Joos M, Lehmann G, Mapelli L, Misiejuk A, Pasqualucci E, Petersen J, Spiwoks R, Tremblet L, Unel G, Vandelli W and Yasu Y 2006 *IEEE Transactions on Nuclear Science* **53** 907–911

[3] Chen J H, Choudhary A, Supinsky D D, DeVries M, Hawkes E R, Klasky S, Liao W K, Ma K L, Mellor-Crummey J, Podhorszki N, Sankaran R, Shende S and Yoo C S 2009 *Computational Science & Discovery* **2** 015001–015031

[4] Geddes C, Toth C, van Tilborg J, Esarey E, Schroeder C, Bruhwiler D, Nieter C, Cary J and Leemans W 2004 *Nature* **438** 538–541

[5] Harris J and STAR collaboration 1994 *Nuclear Physics A* **566** 277 – 285

[6] Chaudhuri S and Dayal U 1997 *ACM SIGMOD Record* **26** 65–74

[7] Devlin B A and Murphy P T 1988 *IBM Systems Journal* **27** 60–80

[8] Inmon W H 1996 *Communications of the ACM* **39** 49–50

[9] Brin S and Page L 1998 *Computer Networks and ISDN Systems* **30** 107–117

[10] Zobel J and Moffat A 2006 *ACM Comput. Surv.* **38** 6

[11] Chan C Y and Ioannidis Y E 1998 Bitmap index design and evaluation *SIGMOD* (ACM press) pp 355–366

[12] O'Neil P 1987 Model 204 architecture and performance *2nd International Workshop in High Performance Transaction Systems, Asilomar, CA* (*Lecture Notes in Computer Science* vol 359) (Springer-Verlag) pp 40–59

[13] Reiss F R 2007 *Data Triage* Ph.D. thesis EECS Department, University of California, Berkeley

[14] Sinha R 2007 *Indexing Scientific Data* Ph.D. thesis UIUC

[15] The HDF Group 2007 HDF5 user guide URL `http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html`

[16] Gosink L, Shalf J, Stockinger K, Wu K and Bethel W 2006 HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices *SSDBM* (IEEE Computer Society Press.)

[17] Childs H, Brugger E S, Bonnell K S, Meredith J S, Miller M, Whitlock B J and Max N 2005 A contract-based system for large data visualization *IEEE Visualization 2005* pp 190–198. Software available at http://wci.llnl.gov/codes/visit/

[18] Rübel O, Prabhat, Wu K, Childs H, Meredith J, Geddes C G R, Cormier-Michel E, Ahern S, weber G H,

Messmer P, Hagen H, Hamann B and Bethel E W 2008 High performance multivariate visual data exploration for extemely large data *SuperComputing 2008 (SC08)* (Austin, Texas, USA) p 51
[19] Schlosser J and Rarey M 2009 *Journal of Chemical Information and Modeling* **49** 800–809
[20] Siqueira T L L, Ciferri R R, Times V C and de Aguiar Ciferri C D 2009 A spatial bitmap-based index for geographical data warehouses *SAC '09* (ACM) pp 1336–1342
[21] Comer D 1979 *Computing Surveys* **11** 121–137
[22] Johnson T 1999 Performance measurements of compressed bitmap indices *VLDB'99* (Morgan Kaufmann) pp 278–289
[23] Wu K, Otoo E and Shoshani A 2006 *ACM Transactions on Database Systems* **31** 1–38
[24] Antoshenkov G and Ziauddin M 1996 *VLDB Journal* **5** 229–237
[25] Wu K, Otoo E and Shoshani A 2001 A performance comparison of bitmap indexes *CIKM* pp 559–561
[26] Wu K, Otoo E and Shoshani A 2002 Compressing bitmap indexes for faster search operations *SSDBM* pp 99–108
[27] Stockinger K and Wu K 2006 *Bitmap Indices for Data Warehouses* in Data Warehouses and OLAP: Concepts, Architectures and Solutions (Idea Group, Inc.) pp 179–202
[28] Wu K, Otoo E and Shoshani A 2001 Compressed bitmap indices for efficient query processing LBNL Tech. Rep. LBNL-47807
[29] Sinha R R and Winslett M 2007 *ACM Trans. Database Syst.* **32** 16
[30] Wong H K T, Liu H F, Olken F, Rotem D and Wong L 1985 Bit transposed files *Proceedings of VLDB 85, Stockholm* pp 448–457
[31] O'Neil P and Quass D 1997 Improved query performance with variant indices *SIGMOD* (ACM) pp 38–49
[32] Wu K, Stockinger K and Shoshani A 2007 Performance of multi-level and multi-component compressed bitmap indexes LBNL Tech. Rep. LBNL-60891
[33] Wu K, Stockinger K and Shosani A 2008 Breaking the curse of cardinality on bitmap indexes *SSDBM'08* pp 348–365 preprint appeared as LBNL Tech Report LBNL-173E
[34] Stockinger K, Wu K and Shoshani A 2004 Evaluation strategies for bitmap indices with binning *DEXA* (Springer-Verlag)
[35] Koudas N 2000 Space efficient bitmap indexing *CIKM* (ACM) pp 194–201
[36] Rotem D, Stockinger K and Wu K 2005 Optimizing I/O costs of multi-dimensional queries using bitmap indices *DEXA* (Springer Verlag)
[37] Rotem D, Stockinger K and Wu K 2005 Optimizing candidate check costs for bitmap indices *CIKM*
[38] Rotem D, Stockinger K and Wu K 2006 Minimizing I/O costs of multi-dimensional queries with bitmap indices *SSDBM* (IEEE)
[39] Wu K, Gu J, Lauret J, Poskanzer A M, Shoshani A, Sim A and Zhang W M 2005 Grid collector: Facilitating efficient selective access from data grids *Proceedings of International Supercomputer Conference In Heidelberg, 2005* LBNL report LBNL-57677
[40] Wu K, Zhang W M, Perevoztchikov V, Lauret J and Shoshani A 2004 The grid collector: Using an event catalog to speed up user analysis in distributed environment *CHEP 2004*
[41] Dean J and Ghemawat S 2004 MapReduce: Simplified data processing on large clusters *OSDI'04*
[42] Sim A and Shoshani A 2007 The storage resource manager interface specification URL `http://sdm.lbl.gov/srm-wg/`
[43] Wu K, Koegler W, Chen J and Shoshani A 2003 Using bitmap index for interactive exploration of large datasets *SSDBM* pp 65–74
[44] Stockinger K, Shalf J, Bethel W and Wu K 2005 Query-driven visualization of large data sets *IEEE Visualization 2005*
[45] Wu K, Otoo E and Suzuk K 2009 *Pattern Analysis & Applications* **12** 117–135
[46] Stockinger K, Bethel E W, Campbell S, Dart E and Wu K 2006 Detecting distributed scans using high-performance query-driven visualization *SC '06* (ACM) p 82
[47] Rübel O, Prabhat, Wu K, Childs H, Meredith J, Geddes C G R, Cormier-Michel E, Ahern S, Weber G H, Messmer P, Hagen H, Hamann B and Bethel E W 2008 Application of high-performance visual analysis methods to laser wakefield particle acceleration data *IEEE Visualization 2008*