Bitmap Indices for Fast End-User Physics Analysis in ROOT

Kurt Stockinger[a], Kesheng Wu[a], Rene Brun[b], Philippe Canal[c]

[a]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

[b]European Organization for Nuclear Research, 1211 Geneva, Switzerland

[c]Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

Most physics analysis jobs involve multiple selection steps on the input data. These selection steps are called *cuts* or *queries*. A common strategy to implement these queries is to read all input data from files and then process the queries in memory. In many applications the number of variables used to define these queries is a relative small portion of the overall data set therefore reading all variables into memory takes unnecessarily long time.

In this paper we describe an integration effort that can significantly reduce this unnecessary reading by using an efficient compressed bitmap index technology. The primary advantage of this index is that it can process arbitrary combinations of queries very efficiently, while most other indexing technologies suffer from the "curse of dimensionality" as the number of queries increases. By integrating this index technology with the ROOT analysis framework, the end-users can benefit from the added efficiency without having to modify their analysis programs. Our performance results show that for multi-dimensional queries, bitmap indices outperform the traditional analysis method up to a factor of 10.

1. Introduction

Typical interactive, end-user physics analysis is an iterative process where data is selected based on specific cuts (*queries*). These queries are often complex and involve several conditions (*dimensions*) such as $npTight < 10$ AND $muonLoose2cm > 5.7$ AND $nTracks > 20$. A common strategy for evaluating these queries is to read all input data from files and then process the queries in memory. However, reading all data values into memory before performing the queries is often very inefficient because most of the data would not be used.

In this paper evaluate the use of bitmap indices for efficient query processing in ROOT. By integrating this index technology with the ROOT analysis framework, the end-users can benefit from the added efficiency without having to modify their analysis programs. We will discuss some implementation details and give a simple use case for performing analysis in ROOT with bitmap indices. Next we compare the performance of bitmap indices with traditional methods. Our measurements show that for multi-dimensional queries, bitmap indices outperform the traditional analysis method up to a factor of 10.

2. Related Work

Bitmap indices are efficient index data structures for speeding up multi-dimensional range queries for read-only data [3,5]. For an attribute with c distinct values, the basic bitmap index [1] generates c bitmaps with N bits each, where N is the number of records in the data set. Each bit in a bitmap is set to 1 if the attribute in the record is of a specific value, otherwise the bit is set to 0. For example, the integer attribute **I** shown in Figure 1 can be one of four distinct values, 0, 1, 2, and 3. The corresponding bitmap index has four bitmaps. Since the value in record 5 is 3, the fifth bit in b_4 is set to 1 and the same bits in other bitmaps are 0.

Bitmap indices are efficient for processing

RID	**I**	bitmap index			
		=0	=1	=2	=3
1	0	1	0	0	0
2	1	0	1	0	0
3	3	0	0	0	1
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. A sample bitmap index where RID is the record ID and **I** is the integer attribute with values in the range of 0 to 3.

multi-dimensional range queries such as "$\mathbf{I} < 2$ and $\mathbf{J} > 3$". The queries are evaluated with bitwise logical operations that are well-supported by computer hardware.

Several bitmap compression methods were studied in [2] to reduce the size of bitmap indices. Note that an efficient bitmap compression scheme not only has to reduce the size of bitmaps but also has to perform bitwise Boolean operations efficiently. More recently a new compression scheme called Word-Aligned Hybrid (WAH) [5] was introduced. It has been shown that even in the worst case, the bitmap indices can be compressed to a size that is comparable with a typical B-tree index. The time required to answer a range query using a compressed bitmap index is in fact optimal. In the worst case, the response time is proportional to the number of hits of the query [5].

The bitmap indices discussed so far encode each distinct attribute value as one bitmap vector. This technique is very efficient for integer or floating point values with low attribute cardinalities. However, scientific data is often based on floating point values with high attribute cardinalities. The work presented in [4] demonstrates that bitmap indices with binning can significantly speed up multi-dimensional queries for high-cardinality attributes.

3. Implementation and Example Usage

We integrated the bitmap indexing technology (developed at Berkeley Lab) into the ROOT-framework to speed up cuts with `TTree::Draw` and `TChain::Draw`. One of the design goals was to integrate the indices in such a way that the end-user only has to make minimal modifications to the analysis code.

3.1. Building Bitmap Indices

A typical example of building bitmap indices within ROOT is as follows.

```
// open ROOT-file
TFile f("data/root/data.root");
TTree *tree = (TTree*) f.Get("tree");

TBitmapIndex bitmapIndex;
bitmapIndex.Init();
char indexLocation[1024] = ‘‘/data/index/’’;
bitmapIndex.ReadRootWriteIndexFile(tree,
  indexLocation);

// build index for two attributes
bitmapIndex.BuildIndex(tree, "npTight",
  indexLocation);
bitmapIndex.BuildIndex(tree, "muonLoose2cm",
  indexLocation);
```

3.2. Simple Analysis with Bitmap Indices

Usually end-user physics analysis is done by performing cuts with TTree::Draw. In this example we show how to do efficient data analysis with bitmap indices, which requires the new class called TBitmapIndex.

```
// open ROOT-file
TFile f("data/root/data.root");
TTree *tree = (TTree*) f.Get("tree");

TBitmapIndex bitmapIndex;
bitmapIndex.Init();
bitmapIndex.Draw(tree, "npTight:muonLoose2cm",
  "npTighta < 10 && muonLoose2cm > 5.7");
```

4. Experimental Results

In this section we evaluate the performance of the bitmap indices in ROOT and compare them with traditional techniques. In particular we measure the performance of multi-dimensional queries both with bitmap indices and `TTreeFormula`. The experiments are based on a data set from the Babar High Energy Physics experiment at Stanford Linear Accelerator Center (SLAC). The original data sets consists of 7.6 million records with some 100 attributes each. For our experiments we randomly chose 10 attributes.

4.1. Size of the Compressed Bitmap Indices

We first measure the sizes of the compressed bitmap indices and compare them with the original, uncompressed data set. The compression factors of the equality-encoded and range-encoded bitmap indices [1] per attribute are shown in Figures 2 and 3. For the two bitmap index strategies we use 1000 and 100 equal-depth bins respectively where each bins has roughly the same number of entries. The compression factors vary between 1 and 13. The total sizes of all ten attributes are shown in Figure 4. We can see that the total size of the equality-encoded bitmap indices is about half of the original data size. The size of the range-encoded bitmap indices, on the other hand, is larger than the base data. Note that typical indices such as the B-tree are often three times larger than the base data. In this sense, the size of the range-encoded bitmap index with 100 bins is still acceptable.

4.2. Query Performance

Next we measure the performance of multidimensional queries. All our experiments are carried out on an Intel Pentium 4 with 2 GB of main memory and a SCSI RAID disk. In order to avoid caching effects during the performance measurements, we flushed the disk cache by unmounting the file system before each query.

In Figures 5 to 8 we show the query response time of 1, 2, 5 and 10-dimensional queries with different *query box* sizes. The query box size is defined as the fraction of the query range with respect to the whole domain space. For instance, for a 1-dimensional query a query box of 0.1 means that the query range covers 10% of the attribute range.

The measurements show that in all case the bitmap index is significantly faster than `TTree::Formala` with a performance improvement up to a factor of 10. We can also see that the range-encoded bitmap index (RE-BMI) performs slightly better than the equality-encoded bitmap index (EE-BMI).

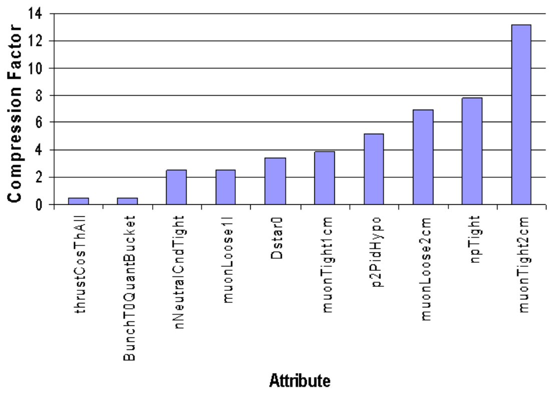

Figure 2. Size of the equality-encoded, compressed bitmap indices per attribute.

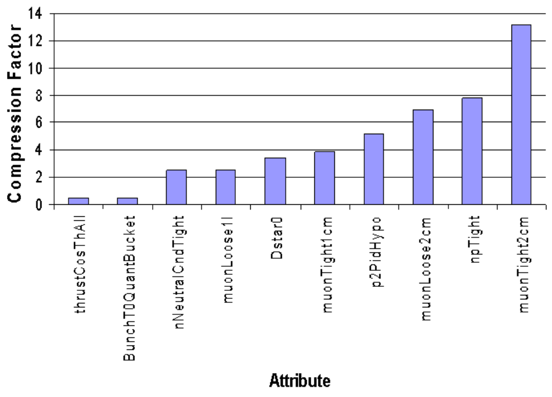

Figure 3. Size of the range-encoded, compressed bitmap indices per attribute.

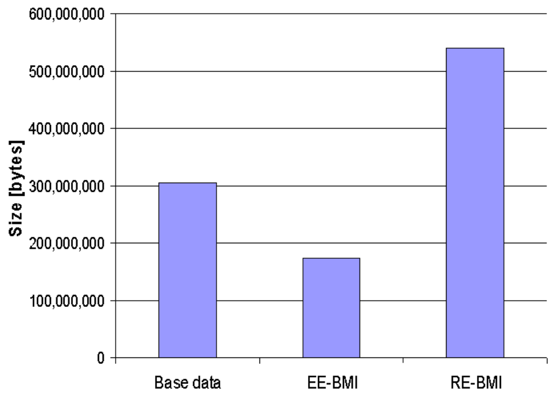

Figure 4. Total size of all compressed bitmap indices. Base data refers to the original, uncompressed data in binary format, EE-BMI refers to equality-encoded bitmap index, and RE-BMI referes to range-encoded bitmap index.

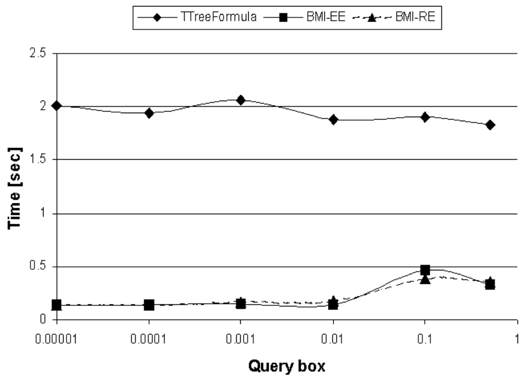

Figure 5. Response time of 1-dimensional queries compared to TTree::Formula.

Bitmap indices with bins provide an additional feature that could be interesting for physicists in

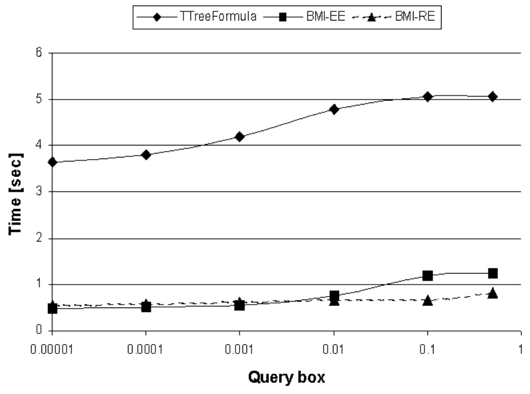

Figure 6. Response time of 2-dimensional queries compared to TTree::Formula.

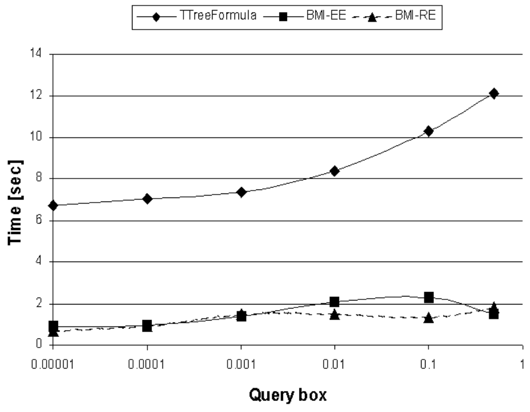

Figure 7. Response time of 5-dimensional queries compared to TTree::Formula.

the initial phase of the interactive data analysis. When using bitmap indices with bins, specific records need to be read from disk in order to check whether they fulfill the query contraint. This is called *Candidate Check* [4]. By omitting

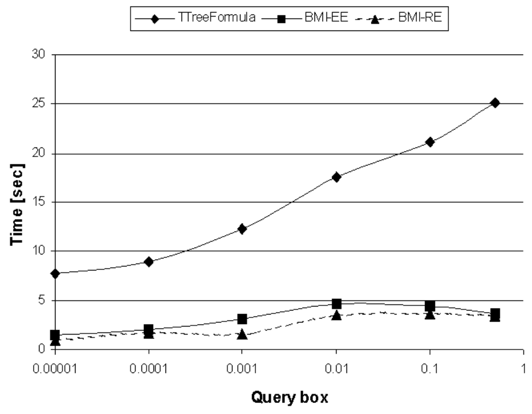

Figure 8. Response time of 10-dimensional queries compared to TTree::Formula.

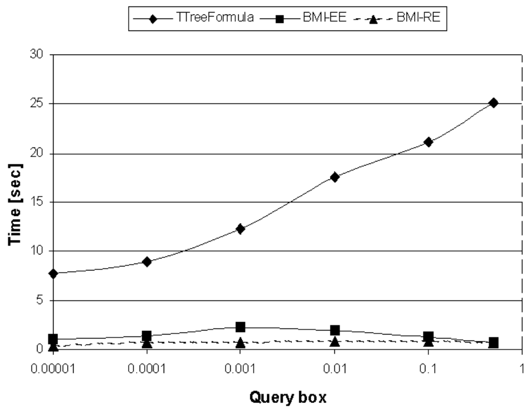

Figure 9. Response time of 10-dimensional queries compared to TTree::Formula. The answers are approximations with 0.1 to 1% errors.

the *Candidate Check* the results are good approximations with error ranges that depend on the number of bins. For instance, for the equality-encoded bitmap index, the error for 1000 bins is 0.1%. For the range-encoded bitmap index with 100 bins, the error is 1%.

Next we measure the performance of multi-dimensional queries with approximate results. Since the characteristics of the query performance is similar for various dimensions, we only show the performance of 10-dimensional queries. As we can see in Figure 9, the performance improvement of approximate queries over `TTree::Formula` is up to a factor of 30.

5. Conclusions

In this paper we discussed the integration of bitmap indices into the ROOT framework. We presented a simple example use case for bitmap indexed accelerated physics analysis and evaluated the performance of the bitmap indices. The results show that for multi-dimensional queries, the bitmap indices outperform the traditional analysis method up to a factor of 10 for exact answers. For approximate queries, we measured a performance gain up to a factor of 30.

REFERENCES

1. C. Y. Chan and Y. E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. In *SIGMOD*, Philadelphia, Pennsylvania, USA, June 1999. ACM Press.
2. T. Johnson. Performance Measurements of Compressed Bitmap Indices. In *International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999. Morgan Kaufmann.
3. P. O'Neil. Model 204 Architecture and Performance. In *2nd International Workshop in High Performance Transaction Systems*, Asilomar, California, USA, 1987. Springer-Verlag.
4. K. Stockinger, K. Wu, and A. Shoshani. Evaluation Strategies for Bitmap Indices with Binning. In *International Conference on Database and Expert Systems Applications (DEXA)*, Zaragoza, Spain, September 2004. Springer-Verlag.
5. K. Wu, E. J. Otoo, and A. Shoshani. On the Performance of Bitmap Indices for High Cardinality Attributes. In *International Conference on Very Large Data Bases*, Toronto, Canada, September 2004. Morgan Kaufmann.