Introduction To FastBit

Outline

- Background
- Use cases
- Bitmap indexes
- Library interface

John Wu
Scientific Data Management
Berkeley Lab

http://sdm.lbl.gov/fastbit

U.S. Department of Energy Contract No. DE-AC02-05CH11231

Context

- Scientific applications are generating enormous amounts of data
- Only a relatively small fraction of data records contain new/unusual/interesting information
- Challenge: find the interesting records quickly and repeatedly
- Solution strategies:
  - Hardware (parallelism): web search engine, hadoop
  - Software: e.g., database management systems, key technologies include indexing, compression and query optimization
  - Software and hardware hybrids, e.g., netezza
- This talk focus on an indexing software designed for scientific applications

Characteristics of Scientific Data

- Data does not change!
  - Once the raw data is captured or computed, most scientific data sets do not change.
- Analyses touch a relatively small number of variables in the data
  - Relevant records are selected based on a few variables
  - Analyses use a small number of variables as input
- Queries are ad hoc in nature, typically involving multi-dimensional range conditions
  - Find collision events where Energy > 200 GeV and 1000 < NumberOfParticles < 2000 and …
  - Find regions in space where temperature > 800 and H_2O_2 concentration > 10^{-5} and …

Common Indexes Not Efficient

Task: searching high-dimensional append-only data with ad hoc range queries

- Most tree-based indexes are designed to support updates
  - Examples: family of B-Trees
  - Efficient for small number of hits
  - Sacrifice some search performance to support updates
- Inverted indexes used in web searching engines not applicable
  - Most scientific data are numbers, while the inverted indexes are for text
- Hash-based indexes are
  - Efficient for finding a small number of records
  - But, not efficient for ad hoc multi-dimensional queries
- Most multi-dimensional indexes suffer from the curse of dimensionality
  - Examples: R-tree, Quad-trees, KD-trees, …
  - Don't scale to high dimensions (< 10)
  - Are inefficient if some dimensions are not queried

FastBit Approach: Bitmap Index

- Bitmap indexes
  - Sacrifice update efficiency to gain more search efficiency
  - Search results from multiple dimensions can be quickly combined
  - Scale linearly with the dimension of a query
- Bitmap indexes may demand too much space
- We solve the space problem by developing an efficient compression method that
  - Reduces the index size, typically 30% of raw data, vs. 300% for some common indexes
  - 10X speedup relative to best known compressed bitmap index
  - Even higher speedup relative to conventional indexes
- We have applied FastBit to speed up a number of DOE funded applications

[Wu, Otoo, Shoshani 2006]

Data Model

- User data are viewed as tables
  - Rows – an observation, variables associated with a mesh point, or other units of data
  - Columns – a variable, field, or other quantities, e.g., temperature, pressure, name, address
- Indexes may be built to accelerate query processing

Data Table

A	B	C	…
0	0.5	AL	
3	0.3	AR	
2	1.2	CA	
1	3.4	CA	
1	0.8	WI	

Bitmaps for A

=0	=1	=2	…
1	0	0	
0	0	0	
0	0	1	
0	1	0	
0	1	0	

Bitmaps for B

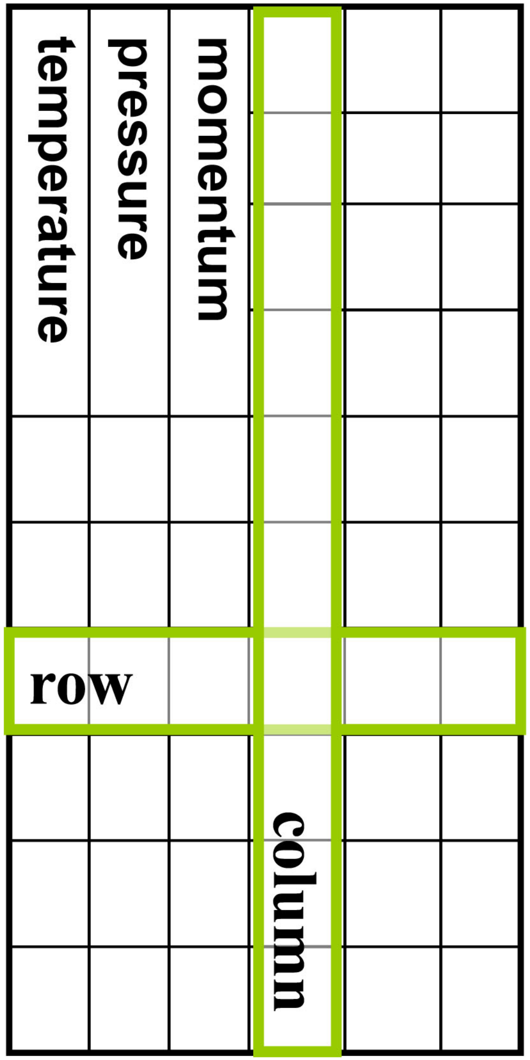
<1	<2	<3	…
1	1	1	
1	1	1	
0	1	1	
0	0	0	
1	1	1	

…

…

FastBit Overview

- Task: given a large collection of data, efficiently locate records satisfying a set of conditions
- Example data – structured data:
  - High-energy physics data – billions of collision events, with hundreds of variables
  - Simulation data on a mesh – each mesh point may be viewed as a record/row, each variable a column
- Example queries:
  - Count how many records where pressure > 1000 and temperature between 500 and 1000
  - Select all records where momentum > …
- FastBit solves this search problem with
  - Column data organization
  - Bitmap index
- FastBit is an award-winning open-source software library
  - R&D100 award (2008)
  - Used in a number of research projects



What FastBit Is Not

- Not a database management system (DBMS)
  - It is much closer to BigTable than to ORACLE
  - Most SQL commands are not supported
- Not a plug-in for a DBMS
  - It is a stand-alone data processing tool
  - No DBMS is needed in order to use FastBit
- Not an internet search engine
  - FastBit is primarily for structured data; internet search engines are for text (unstructured) data
- Not a client-server system
  - We have used FastBit in server programs, but by itself, it is not a client-server system

Introduction To FastBit

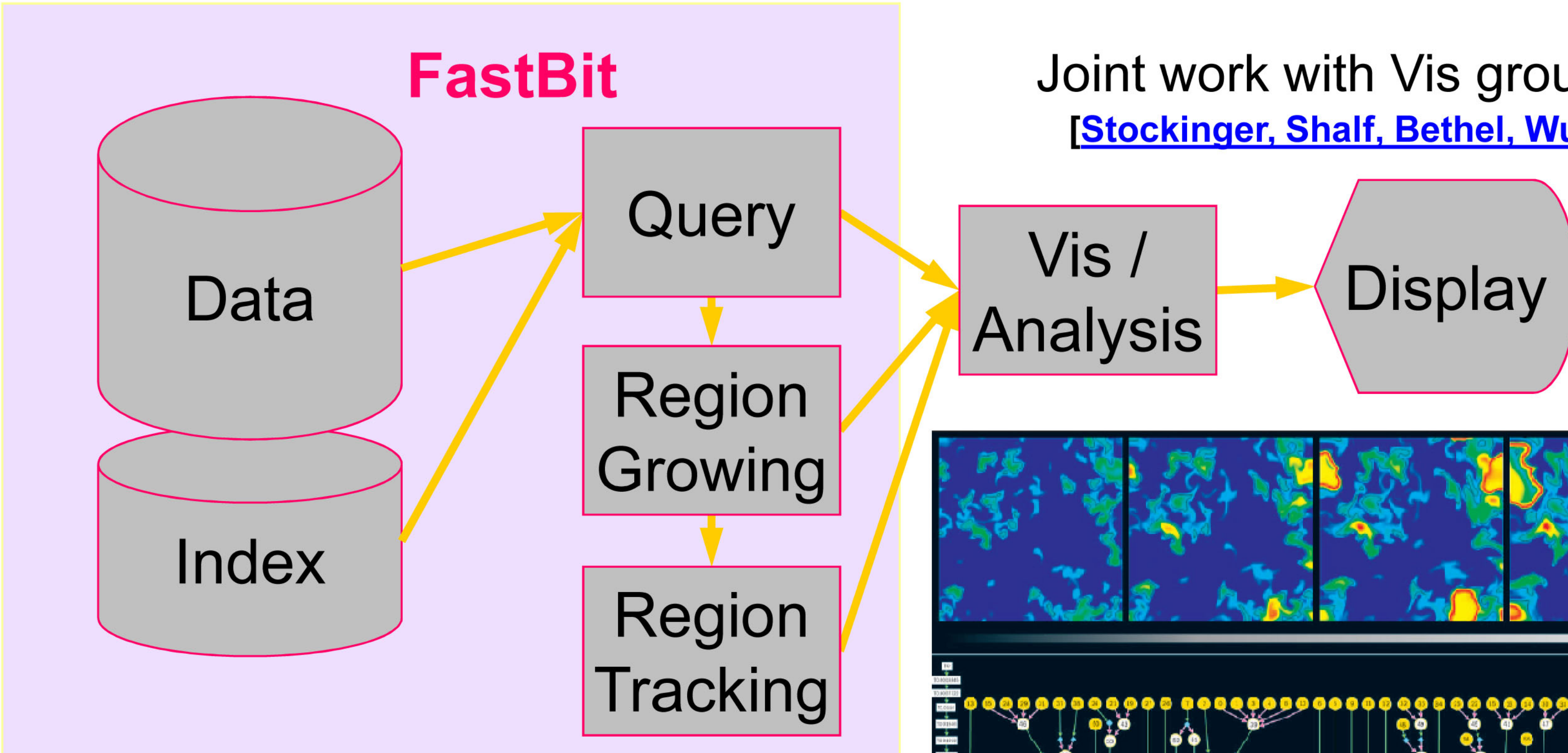
Outline

- Background
- Use cases
- Bitmap indexes
- Library interface

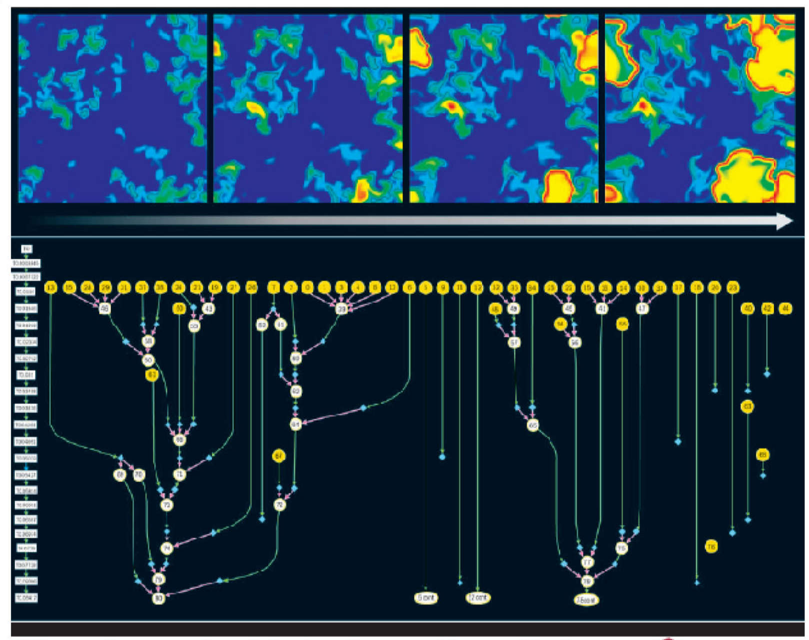
John Wu
Scientific Data Management
Berkeley Lab

http://sdm.lbl.gov/fastbit

Use Case 1: Query Driven Visualization



Joint work with Vis group
[Stockinger, Shalf, Bethel, Wu 2005]

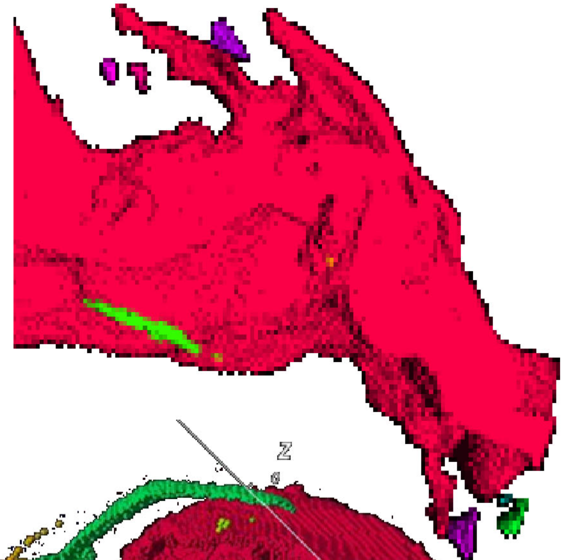


Evolution of ignition kernels (data from Sandia)
[Wu, Koegler, Chen, Shoshani 2003]

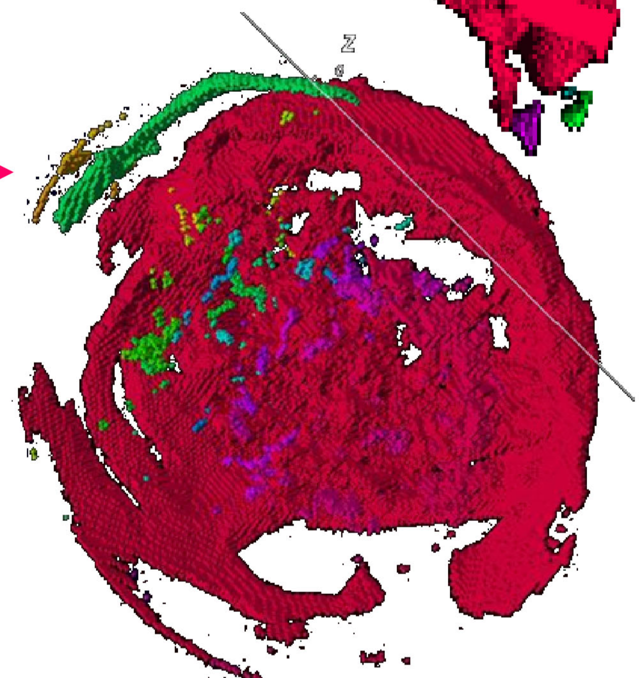
FastBit provides
- Efficient subsetting
- Efficient region-growing
- Efficient region tracking

Query Drive Visualization: Examples

- Find the ignition kernels in a combustion simulation
  - Find regions in space where temperature > 800 and H_2O_2 concentration > 10^{-5} and …



- Track a layer of exploding supernova
  - Tracking regions in space where pressure gradient > 10000 and 2000 < density < 3000 and …

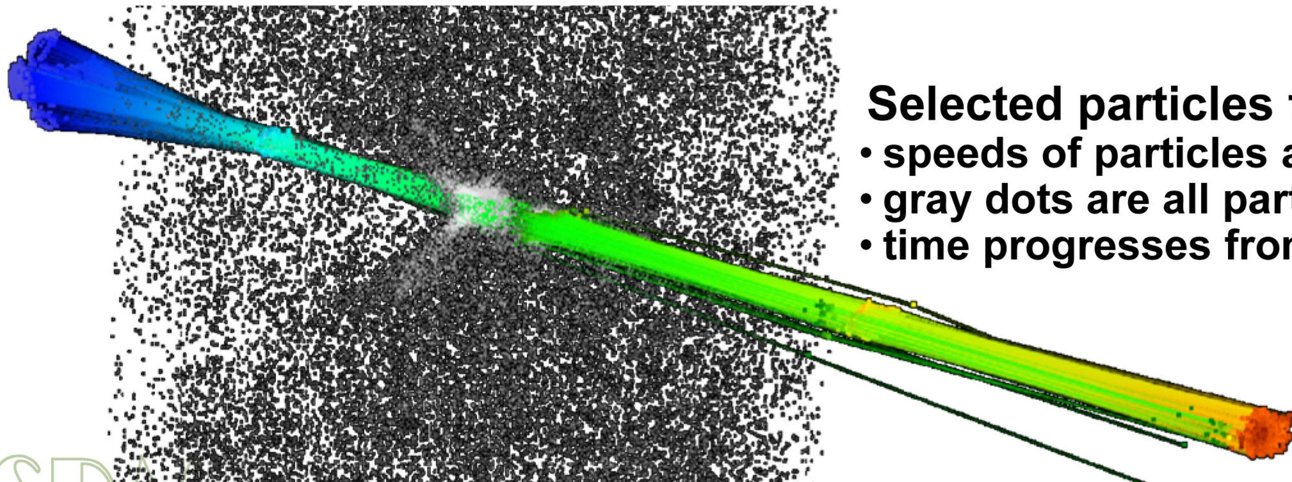


[Stockinger, Shalf, Bethel, Wu 2005]

Use Case 2: Particles in Laser Wakefield

FastBit indexes track particles in Laser Wakefield Accelerator Simulation 3 orders of magnitudes faster than previous methods

- To study the acceleration process, one selects the particles with the highest speed at the end of the simulation
- Use FastBit indexes to directly to access these particles in earlier time steps with the same identifiers, instead of brute-force comparisons
- Hundreds of millions of particles are simulated, but only tens of thousands of particles are highly accelerated, therefore speedup is orders of magnitudes

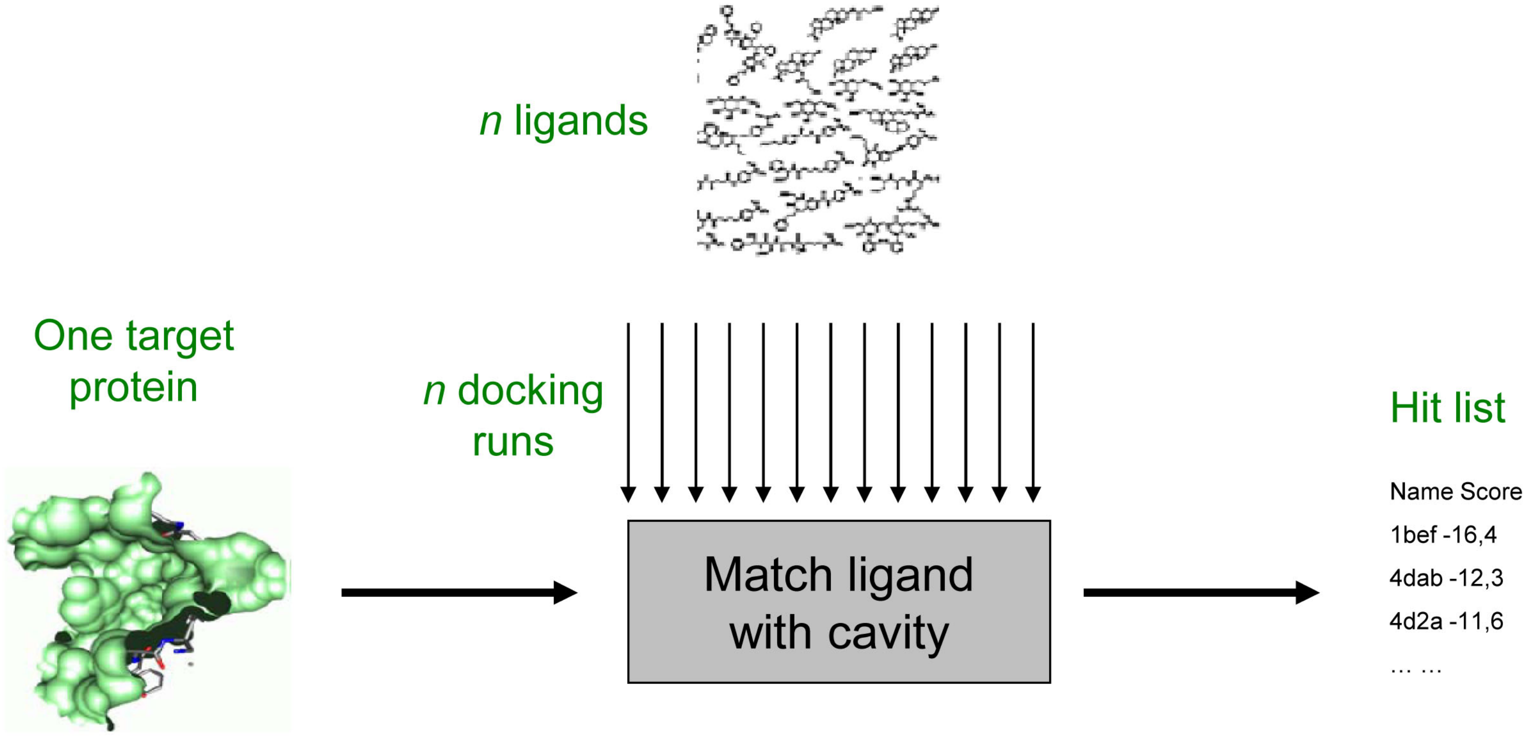


[Rubel et al. 2008]

Selected particles from different time steps
- **speeds of particles are colored from blue to red**
- **gray dots are all particles from one time step**
- **time progresses from left to right**

Use Case 3: Molecular Docking

- **Jochen Schlosser** [schlosser@zbh.uni-hamburg.de]
  Center for Bioinformatics, University of Hamburg
- **Application:** Structure-based virtual screening (ACS Fall 2007, JCIM 2009)

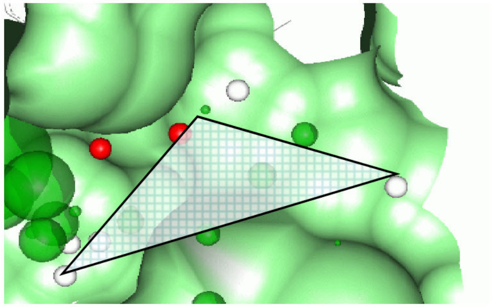


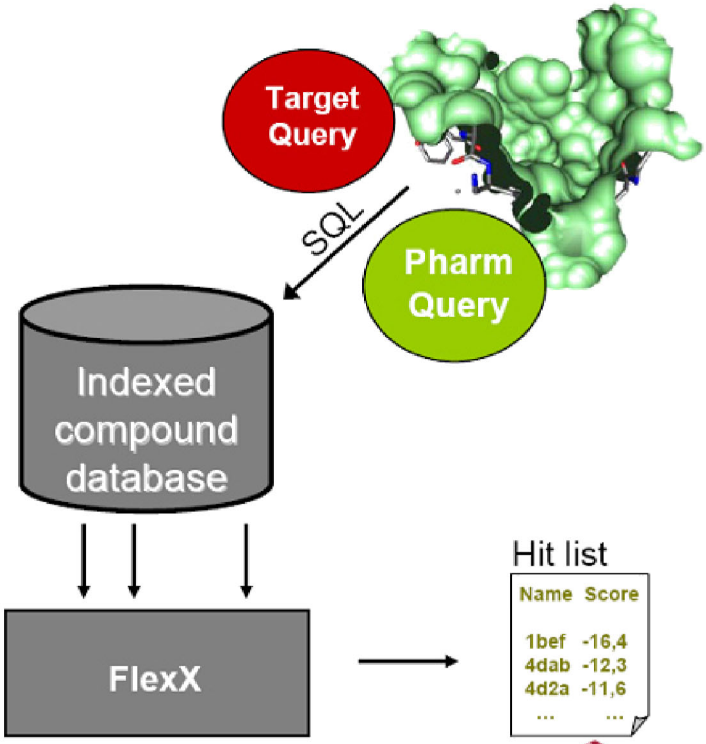
Standard approach: match every ligand with every target protein
New approach: using FastBit indexes to avoid brute-force matching

Use of FastBit for Molecular Docking

Method

- **Describe shape of ligand with triangle geometry**
  - Types of interaction centers
  - Triangle side lengths
  - Interaction directions
  - 80 bulk dimensions
- **Receptors**
  - Receptor descriptors are generated similarly
  - Using complementary information where necessary
- **Use of pharmacophore constraints on receptor triangles**
  - Reduces number of queries
  - Improved query selectivity because the pharmacophore tends to be inside the protein cavity



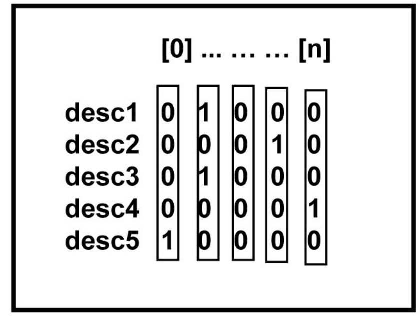


Use of FastBit for Molecular Docking

Method

- Indexing system
  - Properties of the problem:
  - Billions of descriptors (~ 1,000 for each ligand)
  - High dimensional query
- Properties of bitmap indexes
  - Well suited for those kind of queries
  - Can be run stand alone
  - Further compression possible
  - FastBit uses compression

attribute(i)



Bitmap index

Results

- TrixX-BMI is an efficient tool for virtual screening with average runtime in sub-second range
- screen libraries of ligands 12 times faster than FlexX without pharmacophore constraints
- **With pharmacophore constraints, speedup 140 – 250**

Introduction To FastBit

Outline

- Background
- Use cases
- Bitmap indexes
- Library interface

John Wu
Scientific Data Management
Berkeley Lab

http://sdm.lbl.gov/fastbit

Basic Bitmap Index

Data values	b_0 =0	b_1 =1	b_2 =2	b_3 =3	b_4 =4	b_5 =5
0	1	0	0	0	0	0
1	0	1	0	0	0	0
5	0	0	0	0	0	1
3	0	0	0	1	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
0	1	0	0	0	0	0
4	0	0	0	0	1	0
1	0	1	0	0	0	0
	A < 2			2 < A		

- First commercial version
  - Model 204, P. O'Neil, 1987
- Easy to build: faster than building B-trees
- Efficient for querying: only bitwise logical operations
  - $A < 2 \rightarrow b_0$ OR b_1
  - $A > 2 \rightarrow b_3$ OR b_4 OR b_5
- Efficient for multi-dimensional queries
  - Use bitwise operations to combine the partial results
- Size: one bit per distinct value per row
  - Definition: Cardinality == number of distinct values
  - Compact for low cardinality attributes, say, cardinality < 100
  - Worst case: cardinality = N, number of rows; index size: N*N bits

Strategies to Improve Bitmap Index

- Compression
  - Reduce the size of each individual bitmap
  - Best known compression method: Byte-aligned Bitmap Code [Antoshenkov 1994], used in Oracle bitmap index
  - Word-Aligned Hybrid (WAH) code trades some disk space for much more efficient query processing
- Encoding
  - Basic equality encoding, in Model 204
  - Multi-component encoding [Chan and Ioannidis 1998]
  - Multi-level encoding
- Binning
  - Equal-width binning, equal-depth binning, …
  - Has to perform candidate check to rule out false positives, time for candidate check dominates the total query response time
  - Order-preserving Bin-based Clustering (OrBiC)

Indexing Option String

- Syntax
  - <binning … /> <encoding … /> <compression … />
- Binning options
  - Basic binning option: linear scale, log scale, equal-weight
  - Examples:
    - <binning none/>
    - <binning nbins=1000/>
    - <binning begin=10, end=20, scale=linear, nbins=10/>
    - <binning precision=2/>
- Encoding options
  - Three basic options: equality, range and interval
  - Combinations:
    - multi-level, e.g., <encoding interval-equality/>
    - multi-component, e.g., <encoding equality ncomp=2/>
- Compression options
  - Public release only supports WAH compression, most users should leave this part out

Indexing Option Suggestions

- Not specifying any option == default option
  - Use the default unless you known something about your data and query
- The following recommendations primarily depends on the column cardinality and the type of query
  - Definition: column cardinality == number of distinct values actually appear in the data partition
- Cardinality < 100:
  - Equality queries: <binning none/> <encoding equality/>
  - Range queries: <binning none/> <encoding interval/>
- Cardinality < 1,000,000 (Nrows/10):
  - Have disk space (index size 2X raw data size):
    <binning none/> <encoding interval-equality/>
- Very high cardinality: <binning none/> <encoding binary/>
- Small number of values to be queried: use them as bin boundaries, treat the number of bins as the column cardinality above

Introduction To FastBit

Outline

- Background
- Use cases
- Bitmap indexes
- Library interface

John Wu
Scientific Data Management
Berkeley Lab

http://sdm.lbl.gov/fastbit

How Do I Use FastBit

- Command-line tools
  - A handful of command-line tools are available to load data, build indexes, and query data
  - But, most likely you will have to write some C/C++ code
- Write your own program using FastBit as a library
  - Two levels of API:
    - Class table
    - Class part + query
  - FastBit is written in C++
    - Other languages may access FastBit through C API

FastBit Native Data Format

- A data table may be split into multiple partitions
- Each partition is stored in a data directory on the file system
- Each column has its own data file (column organization)
- Each column has its own bitmap index
- Each data partition has a metadata file describing the partition (example on the right)

```
BEGIN HEADER
DataSet.Name=testData
Number_of_rows=1000000
Number_of_columns=6
Table_State=1
index = <binning none/>
END HEADER

BEGIN Column
name=i9
description=integers 0, 1, ..., and 9
data_type=Int
index = <encoding range/>
END Column
…
```

FastBit Command-Line Tools

- All source code for these tools are in examples directory
- Ardea: convert text version of the data records into FastBit raw binary data format – an operation common known as "load"
  - Ardea –d output-dir –t text-file –m columnname:type
- Ibis: query existing data
  - Ibis –d data-dir –q "select c1,c2 where c3 > 5 and c4 < 6"

Software Layering

- Abstract view: ibis::table and ibis::tablex
  - A table is immutable; to add new records, use tablex
  - A query (through function select) produces another table
  - Additional functions include: build indexes, get conditional histograms, get column values, …
- Concrete view: ibis::part and ibis::query
  - Each part (partition) is vertically organized
  - An index for a column of a partition is built in memory
  - A query on partition produces a compressed bitmap representing the rows satisfying the specified conditions

Ingesting Data

- Uses `ibis::tablex`, excerpt from `examples/ardea.cpp`

```
                                   // create a tablex object
ibis::tablex* ta = ibis::tablex::create();
                                // parse the metadata string
ta->parseNamesAndTypes(metadata.c_str());
                  // read CSV file, store content in memory
ierr = ta->readCSV(csvfiles[i], nrpf, del);
// write the content from memory to the named directory
ierr = ta->write(outdir, “name”, “some description");
```

Simple Queries

- Uses ibis::table, excerpt from examples/thula.cpp

```
                // create a table object from a directory name
ibis::table *tbl = ibis::table::create("directory-name");
                         // a selection creates another table
ibis::table *res = tbl->select("select clause", "where clause");
   // create a cursor for row-wise access to the results
ibis::table::cursor *csr = res->createCursor();
                // fetch the next row and dump it to std::cout
while (0 == csr->fetch())
   csr->dump(std::cout);
```

Low-Level Query Functions

- Uses `ibis::part` and `ibis::query`, excerpt from `examples/rara.cpp`

```
        // construct a data partition from a directory
ibis::part apart(argv[1], static_cast<const char*>(0));
    // create a query object with the current user name
ibis::query aquery(ibis::util::userName(), &apart);
     // assign the query conditions as the where clause
int ierr = aquery.setWhereClause(argv[2]);
                              // select columns to print
ierr = aquery.setSelectClause(sel.c_str());
                                   // evaluate the query
ierr = aquery.evaluate();
                            // print the selected values
aquery.printSelected(std::cout);
```

Histogram Functions

- Conditional histograms are commonly used in data analyses
  - Count the number of events collected every hour for all events from a particular day (1-D)
  - Count the number of network connection attempts per minute per destination port for a specific duration of time (2-D)
- Class ibis::part also has a set of functions to compute histograms
  - get1DDistribution
  - get2DDistribution
  - get3DDistribution
  - May use regular bins or adaptive bins
  - May be weighted by another variable
- FastBit uses indexes to reduce the amount of data accessed and speeds up the histogram computations

Querying Long List of Values

- A useful functionality for tasks such as tracking particles with ids and do-it-yourself joins
- Directly construct a query expression with binary values; bypassing the string parsing
  - Place the list of values in an std::vector<double>, say, vals
  - Constructor an ibis::qDiscreteRange
  - `ibis::qDiscreteRange dr("column-name", vals);`
- Set the where clause by directly using the query expression
  - `aquery.setWhereClause(&dr);`
- Evaluate the query as usual
  - `aquery.evaluate();`

Index Sizes to Expect

- Indexes are built for one column and one partition at a time
- The maximum size of an index is primarily determined by three parameters: the number of rows N, the number of bitmaps used B, and the bitmap encoding used.
- The range and interval encoded indexes are not compressible in the worst case, therefore their sizes are N * B bits
- Under the equality encoding, for a binned index, B is the number of bins, otherwise the number of bitmaps is the number of distinct values (i.e., column cardinality)
  - For small B, say, B < 100, N * B bits are needed because bitmaps are likely not compressible
  - For B < N / 10, the common case, index size is about 2 N words
- For columns with extremely high cardinality, use binary encoding, which requires log B bitmaps and N * log B bits

Updating Data and Indexes

- Most efficient way to add new records is to add a partition to an existing table
- Modifying an existing row must be implemented as a deletion following by an append
- Updating an index on a partition will cause a whole new index to be written, which can take a long time compared to the time to answer a query
- To improve response time, such updates are allowed to be delayed, presumably till the system is no longer busy

Parallelism

- Using ibis::part and ibis::query, each parallel processing element could work on one data partition
  - Additional code required to synthesize the final result
- Additional parallelism can come from having each processor answer a part of a query
  - For a query involving "a > 2 and b < 3", process the condition involving a and b on two separate threads or processors
  - Require additional code to combine the partition results
- Prefer to have more partitions than the number of processors to improve load balancing
- The original version of FastBit was a CORBA server program
  - Current code were the core of the multithreaded server, minus the CORBA functions
  - All existing code should be thread-safe

ANY QUESTIONS?

More information at

http://sdm.lbl.gov/fastbit

Please join the FastBit mailing list

https://hpcrdm.lbl.gov/cgi-bin/mailman/listinfo/fastbit-users