In-Situ Bitmaps Generation and Efficient Data Analysis based on Bitmaps

Yu Su
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
su1@cse.ohio-state.edu

Yi Wang
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
wayi@cse.ohio-state.edu

Gagan Agrawal
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
agrawal@cse.ohio-state.edu

ABSTRACT

Neither the memory capacity, memory access speeds, nor disk bandwidths are increasing at the same rate as the computing power in current and upcoming parallel machines. This has led to considerable recent research on *in-situ* data analytics. However, many open questions remain on how to perform such analytics, especially in memory constrained systems. Building on our earlier work that demonstrated bitmap indices (bitmaps) can be a suitable summary structure for key (offline) analytics tasks, this paper develops an in-situ analysis approach that performs data reduction (such as time-steps selection) using just bitmaps, and subsequently, stores only the selected bitmaps for post-analysis. We construct compressed bitmaps on the fly, show that many kinds of in-situ analyses can be supported by bitmaps without requiring the original data (and thus reducing memory requirements for in-situ analysis), and instead of writing the original simulation output, we only write the selected bitmaps to the disks (reducing the I/O requirements). We also demonstrate that we are able to use bitmaps for key offline analysis steps. We extensively evaluate our method with different simulations and applications, and demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

H.3.1 [**Information Systems**]: CONTENT ANALYSIS AND INDEXING—*Indexing Methods*; H.1.1 [**Information Systems**]: SYSTEMS AND INFORMATION THEORY—*Information Theory*

Keywords

in-situ analysis; bitmaps; data reduction; time-steps selection; correlation analysis

1. INTRODUCTION

Data-driven discovery from scientific simulations is facing a major and *disruptive* challenge, due to the recent and emerging trends in high performance computing systems. Specifically, there is a shift towards architectures where memory and I/O capacities are not keeping pace with the increased computing power. There are many reasons for this constraint in high performance systems where science and engineering simulations are typically run. Most critically, the need for providing high computational power in a cost and power effective fashion is driving architectures with limited memory and severe data movement limitations [18, 24].

Examples of such designs can be seen from the architectures of accelerators and coprocessors like GPUs and Intel MIC, which have a large number of cores but only a small amount of memory per core. In fact, a detailed analysis of the supercomputers in the top 500 list has shown a clear decrease in the memory per Gflops in recent years [19]. A related issue is that the data movement costs have become the dominant consideration in design and operation of powerful parallel machines, and all indications are that they will be the bottleneck in the future. It needs to be noted that the memory-constrained powerful HPC systems are in contrast to the trend towards large main memory and memory-resident databases or analytics seen in 'big data' market. Servers with massive memory do not provide computing power in a cost and power effective fashion. Since the first order need of science and engineering simulations is large-scale computing, it is not a surprise that they are being increasing executed on systems with coprocessors and accelerators, and thus are facing memory and data movement bottlenecks.

Driven by the memory and data movement considerations, we can see that the only option for data-driven analysis of an engineering simulation on emerging (cost and power effective) systems involves: 1) Aggressive *reduction* of data soon after it is generated (so as to reduce the memory requirements for the next step), 2) Analyses performed in real-time over the reduced or summarized data, and possible further data reduction, 3) Long-term storage and/or movement of only the most critical and summarized data for future analyses, and 4) Aggressive analyses, visualization, and exploration, but using only the summarized data. The first two steps are referred to as *in situ* data reduction and data analysis, respectively [3, 16, 17, 20, 21, 28, 34].

Such in situ analysis, however, involves many open challenges. On one hand, memory and data movements considerations dictate that any summary structure be very compact, analyses be extremely memory efficient, and only a small fraction of the data generated be stored in the persistent memory or moved to a different device. On the other hand, it is also important that data analysis does not lose out on salient features – otherwise, the entire advantage of simulating the phenomenon at a high spatial and/or temporal granularity can be easily lost. Scientists often perform data reduction based on ad-hoc approaches, typically relying on their own insights into the simulation. However, in the process, they risk losing out on new insights into the phenomenon that can only be extracted by a more systematic analysis of data.

In our work, we propose a novel method that utilizes *bitmap index (bitmaps)* as a summarization of the data and showed that many kinds of analyses can be supported by bitmaps without requiring the original dataset. This work develops on top of our recent work

HPDC'15, June 15–20, 2015, Portland, Oregon, USA.

http://dx.doi.org/10.1145/2749246.2749268.

ID	Value	e_0	e_1	e_2	e_3	i_0	i_1
		=1	=2	=3	=4	[1, 2]	[3, 4]
0	4	0	0	0	1	0	1
1	1	1	0	0	0	1	0
2	2	0	1	0	0	1	0
3	2	0	1	0	0	1	0
4	3	0	0	1	0	0	1
5	4	0	0	0	1	0	1
6	3	0	0	1	0	0	1
7	1	1	0	0	0	1	0
Dataset		Low Level Indices				High Level Indices	

Figure 1: An Example of Bitmap Index

that demonstrated how bitmaps can be used as a compact summary of data and lead to efficient implementations of key (offline) analytics tasks [30]. This paper demonstrates how bitmaps can be used efficiently for in-situ analytics. Specifically, after the data is simulated in memory, we generate bitmaps, and subsequently, a number of different analysis steps are performed on bitmaps. Then, instead of writing the data, we only write the bitmaps (much smaller in size) back to disk, which greatly improves the I/O efficiency. We have also demonstrated how to use bitmaps for both online (in-situ) analysis (e.g., time-steps selection which focuses on selecting a subset of time-steps containing more *important* information out of simulated time-steps [22, 31, 35, 36]) and offline analysis (e.g., correlation mining which automatically finds data subsets with high correlations [5]) efficiently.

We have extensively evaluated our approach. We first show that although our method takes extra time for bitmaps generation, the time requirements for the entire in-situ analysis phase are lowered in most of the cases (the speedup is from 0.79x to 3.28x). This is because our method has much smaller online analyses and I/O time cost, and we also improved the bitmaps generation efficiency utilizing multi-core and many-core architectures. Besides efficiency improvement, another advantage of our method is that it has much smaller memory consumption because we only keep bitmaps (much smaller in size) instead of the original dataset in memory. This way, we can hold more time-steps in memory for more accurate online analysis. Second, we design different core allocation strategies and show how total time cost is affected by different strategies. Third, we show that the efficiency of in-situ bitmaps generation and online analysis can be improved in a parallel in-situ environment. Fourth, we show that our method can improve the efficiency for offline analyses such as correlation mining. Finally, we compared the in-situ sampling method with our in-situ bitmaps method and show that while our method does not incur any information loss for time-steps selection, in-situ sampling method incurs an average of 21.03% to 58.73% information loss depending on the sampling percentage.

2. IN-SITU BITMAPS GENERATION

This section first gives the background of bitmap index (bitmaps) and then describes our in-situ bitmaps generation method.

2.1 Background: Bitmap Index

An indexing method provides an efficient way to support value-based queries. Various methods have been extensively researched and used in the context of relational databases. Bitmap indexing, which utilizes the fast bitwise operations supported by the computer hardware, has been shown to be an efficient approach, and has been widely used in scientific data management [26, 43]. In particular, recent work has shown that bitmap index can help support efficient querying of scientific datasets stored in native formats [7, 29].

Figure 1 shows an example of a bitmap index. In this simple example, the dataset contains a total of 8 elements with 4 distinct values. The *low-level* bitmap index contains 4 bitvectors, where each bitvector corresponds to one value. The number of bits within each bitvector is the same as total number of elements in the dataset. In each bitvector, a bit is set to 1 if the value for the corresponding data element's attribute is equal to the *bitvector value*, i.e., the particular distinct value for which this vector is created. The *high-level* bitmap index can be generated based on either the value intervals or value ranges. From Figure 1, we can see two *high-level* bitvectors are built based on value intervals.

This simple example only contains integer values. Bitmap indexing also has been shown to be an efficient method for floating-point values [42]. For such datasets, instead of building a bitvector for each distinct value, we can first group a set of values together (*binning*) and build bitvectors for these bins. This way, the total number of bitvectors is kept at a manageable level.

From the example we can also see that the number of bits within each level of bitmap index is $n \times m$, where n is the total number of elements and m is the total number of bitvectors. This can result in sizes even greater than the size of the original dataset, causing high time and space overheads for bitmaps creation, storage, and query processing. To solve this problem, *run-length compression* algorithms such as Byte-aligned Bitmap Code (BBC) [4] and Word-Aligned Hybrid (WAH) [41] have been developed to reduce the bitmaps size. The main idea of these approaches is that for long sequences of 0s and 1s within each bitvector, an encoding is used to count the number of continuous 0s or 1s. Such encoded counts are stored, requiring less space. Another property of the run-length compression methods is that it supports fast bitwise operations without decompressing the data.

2.2 Summarizing Spatial Data with Bitmaps

The following observations can be made with respect to bitmap index and their suitability as a summary structure for multi-dimension arrays. There is no binning (and thus no loss of precision) with respect to dimensional attributes, unlike almost any other method, such as a histogram. This turns out to be a very important advantage, especially for any application where spatial precision is critical. The value distribution is also maintained (though this is also true for histograms). At the same time, because each point is represented by a single bit, compression is used, continuity of dimensional attributes is exploited, and due to the hardware support for bit operations, we can achieve space and time efficiency. Note that like other methods discussed above, there is binning with respect to value-based attributes, but it seems unavoidable when space efficiency is important.

The other advantages of bitmaps are as follows:

- Bitmaps are much smaller in size compared with the original dataset. In most of the cases, the size of bitmaps is less than 30% in size compared with the original dataset, which improves both data I/O and memory usage.
- Many kinds of analyses can be executed purely using bitmaps without the original dataset. In our previous work, we demonstrated that approximate data aggregation, data spatial join, correlation query, incomplete data analysis and subgroup discovery can be supported using bitmaps without touching the original dataset [2, 30, 38, 39]. In this work, we will further show that time-steps selection and correlation mining can be supported using bitmaps.
- Bitmaps can be generated efficiently in an in-situ setting with acceleration using multiple cores. More importantly, we observe that newer architectures have a large number of cores, which can be used to generate bitmaps without very high overheads. Moreover, the cost of bitmaps generation is easily offset by reduction in data movement times.

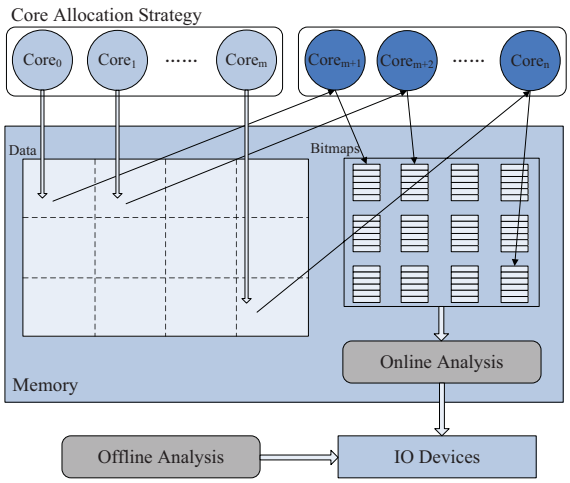


Figure 2: In-Situ Bitmaps Generation

2.3 In-Situ Bitmaps Generation

Recently, several others efforts [16, 20] demonstrate that the bitmap index generation speed can be greatly improved in an in-situ setting. Compared with these works, the main contribution of our work is that we design a WAH-based in-place bitmaps compression algorithm which has very small memory cost and hence is suitable for in-situ setting. We also propose different core allocation strategies for parallel bitmaps generation and discuss the advantage and disadvantage of each strategy.

Multi-Core Processing and Core Allocation: Figure 2 shows the process of generating distributed bitmaps from the data output using an ongoing simulation, and using multiple cores on a single node. The same algorithm can be applied if the data is simulated in a cluster environment where each machine simulates a portion of the data. A *Core Allocation Strategy* is pre-defined, and is used to decide how to allocate cores for simulation and bitmaps generation. Using the strategy, a set of cores ($core_0$ to $core_m$) is assigned for data simulation, while the remaining cores are responsible for bitmaps generation. Each simulation core will simulate a portion of the data and write the data into memory. After a time-step of data is ready, the data will be logically partitioned into $(n - m)$ sub-blocks based on a *pre-defined* partitioning algorithm. Then, each core among the cores assigned for bitmaps generation will access a sub-block of the data in memory and generate compressed bitvectors. This way, the bitmaps generation is performed in parallel without having any dependency among different cores. After bitmaps are generated, the original data can be discarded because bitmaps contain enough information for users to do certain kinds of online (e.g., time-steps selection) and offline analysis (e.g. correlation mining), which will be described in the next two sections.

In our work, two cores allocation strategies (*Shared Cores* and *Separate Cores*) are designed. The *Shared Cores* method assigns all the available cores for both data simulation and bitmaps generation. Each time after one time-step of data is generated (using all cores), the simulation program will be paused and all the cores will be switched for bitmaps generation. After that, the simulation continues to generate next time-step data when the bitmaps for current time-step are ready. The *Separate Cores* method divides the available cores into two separated sets. One set is always used for data simulation and another is responsible for bitmaps generation. In this case, the data simulation and bitmaps generation is performed in parallel, and a data queue is shared between simulation and bitmaps generation. Each time when a new time-step data is simulated, it will be added to the tail of the data queue if the queue is not full (the queue size is limited by the memory capacity). And if the data queue is not empty, the data will be dequeued from the head of the queue and the cores for bitmaps generation will generate bitmaps based on the data. Using this method, the number of cores allocated between simulation and bitmaps generation becomes very important. Ideally, the average data simulation speed should be close to the average bitmaps generation speed after the core allocation. In our work, we use Equations 1 and 2 to decide core allocation. For each simulation, we first use an initial set of cores to run simulation and bitmaps generation and calculate the average simulation time ($Time_{simulate}$) and bitmaps generation time $Time_{bitmap}$. Then the number of cores of allocation depends on the ratio between $Time_{simulate}$ and $Time_{bitmap}$.

$$\text{Core}_{simulate} = \text{Core}_{total} \times \frac{\text{Time}_{simulate}}{(\text{Time}_{simulate} + \text{Time}_{bitmap})} \quad (1)$$

$$\text{Core}_{bitmap} = \text{Core}_{total} - \text{Core}_{simulate} \quad (2)$$

Algorithm 1: Generate_Bitmaps($Data, DataSize, BinNum$)

```
1:  id = 0;
2:  allocate space for Segments with size BinNum;
3:  allocate space for Result with size BinNum;
4:  for i = 0; i < DataSize; i+ = 31 do
5:     initialize elements in Segments to 0;
6:     for j = 0; j < 31 AND j + i < DataSize; j+ = 1 do
7:        VectorID = MapValueToID(Data[id + +]);
8:        Segments[VectorID] = Segments[VectorID] | (1UL ≪
          (30 − j));
9:     end for
10:    for j = 0; j < BinNum; j+ = 1 do
11:       &LastSeg = Result[j].back()
12:       if Segments[j] == 0x7FFFFFFF then
13:          if (LastSeg AND 0xC0000000) == 0xC0000000 then
14:             LastSeg+ = 31;
15:          else
16:             Result[j].push_back(0xC000001F);
17:          end if
18:       else if Segments[j] == 0 then
19:          if (LastSeg AND 0xC0000000) == 0x80000000 then
20:             LastSeg+ = 31;
21:          else
22:             Result[j].push_back(0x8000001F);
23:          end if
24:       else
25:          Result[j].push_back(Segments[j]);
26:       end if
27:    end for
28: end for
29: Return Result
```

Online Compression: For in-situ analysis, besides the efficiency, another important factor is the memory limitation. We do not want bitmaps to occupy a large amount of memory and thus affect the simulation process. From Subsection 2.1, we know that bitmaps before compression can require more memory than the original data. Hence, it is certainly unacceptable that we first generate all uncompressed bitvectors and then do a one-time compression. The problem, however, is that bitmap compression is normally based on identifying series of 0's or 1's, something that is hard if only small segments are available.

We address this problem through a novel algorithm. The basic idea of this algorithm is to fetch data segment by segment, generate bitvectors for each segment, and merge them into existing compressed bitvectors, while only needing to scan the data once. Moreover, because our goal is to only keep bitmaps instead of data for certain kinds of analysis, we can keep freeing the memory of the data we have already scanned and use it to hold bitmaps we generated. This way, the available memory actually

keeps increasing as bitmaps are generating. Algorithm 1 shows the pseudocode of this method. The variable id is the index to iterate through the data, the variable $Segments$ contains the uncompressed bitvectors for the current segment, and the variable $Result$ contains the compressed bitvectors for all previous segments. The for loop in the line 4 shows that we generate the bitvectors segment by segment (we use WAH compression, where every 31 elements forms a segment). Line 5 to line 9 shows the process of generating uncompressed bitvectors ($Segments$) for current segment. $Segments$ is a 2D bit array (the first dimension is the number of bitvectors, the second dimension is 32 bits stored as one integer) and initially every bit is set to 0, as shown in line 5. Line 7 maps each element in current segment into corresponding bins (bitvectors) based on its value. Line 8 sets the corresponding bit of the mapped bitvector to 1 based on element position. Line 10 to line 28 shows the process of merging current uncompressed bitvectors ($Segments$) into existing compressed bitvectors ($Result$). Line 11 fetches the last compressed bitvector unit from $Result[j]$. If current $Segments[j]$ contains all 1-bits ($0x7FFFFFFF$), and if the previous compressed bitvector unit is a fill word containing 1-bits ($0xC0000000$), we update the last compressed unit by adding 31 new 1-bits, as shown from line 12 to line 14. Otherwise, we add a new fill word ($0xC000001F$) containing 31 bits into $Result[j]$, as shown from line 15 to line 17. Line 18 to line 23 shows the merging process when $Segments[j]$ is a fill word containing all 0-bits (0), and the logic is similar as 1-bit fill word. If $Segments[j]$ contains a mix of 0 and 1 bits, we treat it as a literal word, and directly add it to the tail of $Result[j]$, as shown from line 24 to 26.

3. ONLINE ANALYSIS USING BITMAPS

Our claim is that bitmaps can be an effective summary of the original data. To this end, we show how some of the key data selection steps can be performed using only bitmaps (and after original data has been discarded). Performing such a step using only bitmaps greatly reduces memory requirements, a critical issue for today's accelerators and upcoming HPC architectures.

The specific data selection step we have implemented using bitmaps is *time-steps selection*. Specifically, for most of the simulations, the data is simulated time-step by time-step. Certain time-steps contain more *important* information over others, and can be one of the representative time-steps for a post analysis step. Thus, the goal of a *time-step* selection algorithm is to find K of the N given time-steps that represent the evolution of the phenomenon.

3.1 Importance Driven Time-steps Selection

Importance driven time-steps selection has been well studied [22, 31, 35, 36, 37]. The key question is how to measure the *importance* of a time-step – usually, the *importance* of a time-step is determined in two aspects: First, the output for the time-step itself may contain a high amount of *information*. Second, the time-step may convey a distinct type of information with respect to the other time-steps. Several correlation metrics from the information theory [6, 8, 32, 33] such as *Earth Mover's Distance*, *Shannon's Entropy*, *Mutual Information* and *Conditional Entropy* are applied to quantify the above two aspects.

The next question is selecting a subset of time-steps given a fixed number of time-steps. Wang *et al* [36] proposed a greedy algorithm, where the main steps are: 1) partition the time-steps into *intervals*, 2) calculate the *correlation* between the previous selected time-step (the one of the previous interval) and each time-step within this interval, and 3) select the time-step with the minimum correlation. Now, the key consideration is generating these partitions. One obvious method will be *fixed length partitioning*, i.e., each partition contains the same number of time-steps. However, more sophisticated method like *information-volume based partitioning* can also be used, where the time-steps within each partition contain the same total amount of accumulated importance values. Note that the above method is a greedy method, and there are

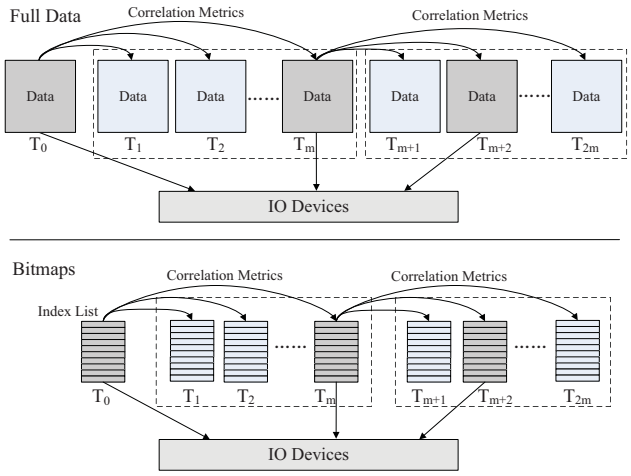


Figure 3: Time-Steps Selection Using Full Data and Bitmaps

other possibilities as well. For example, Tong *et al* [31] proposed a method that uses dynamic programming. Though bitmaps can be used to accelerate computation of almost any such algorithm, we have implemented a greedy algorithm, because efficiency is the most important consideration for us.

Figure 3 shows the process of supporting time-steps selection using both *full data* and *bitmaps*. We first look at the *full data* method. From the figure we can see that, there is a total number of $2m + 1$ time-steps, which are divided into three intervals. The first interval only contains one time-step (T_0), while the second and third interval contain m time-steps each. Initially we always choose the first time-step as the pre-selected time-step. To find the time-step in the second interval with minimum correlation with respect to the first time-step, we need to calculate different correlation metrics between the pre-selected time-step T_0 and each time-step in the second interval (T_1 to T_m). After the calculation, T_m contains the minimum correlation with respect to T_0. Hence, we only keep T_m for the second interval and discard others. Then, we use T_m as the preselected time-step and continue to select the time-step with minimum correlation in the third interval, and continue the process. This way, we are able to keep time-steps with maximum information with respect to each other. If we look at the *bitmaps* method, we can see that the calculation process is similar. However, instead of using the original data which is big in size, all the correlation metrics can be calculated using bitmaps. Because bitmaps is much smaller in size, this method has much smaller memory cost. And in the following subsection, we will also show that the calculation of different correlation metrics can be performed more efficiently using bitmaps.

The algorithm we have accelerated using bitmaps uses two important correlation metrics, the *Earth Mover's Distance* and *Conditional Entropy*. The calculation of *Conditional Entropy*, in turn, is based on two other metrics, which are *Mutual Information* and *Shannon's Entropy*. To be able to describe how these metrics can be computed using bitmaps, we first give mathematical definitions of these metrics, and how they are computed using original data.

Earth Mover's Distance: The earth mover's distance (EMD) is the measure of the distance between two probability distributions over a region D – it can be viewed as the cost of changing one distribution of the data to another distribution of the data. Let us say that we are looking at the difference of distribution between two time-steps A and B. Then, *EMD* is calculated by first dividing elements of attribute A and B into bins based on values (the binning range of different time-steps should be the same), then checking how many elements are different within each bin between A and B, and finally, taking a cumulative sum of these differences together,

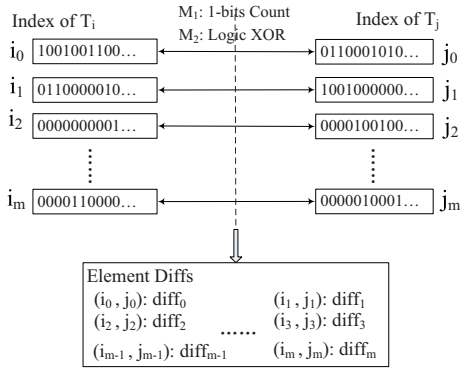


Figure 4: Calculating Earth Movers Distance Using Bitmaps

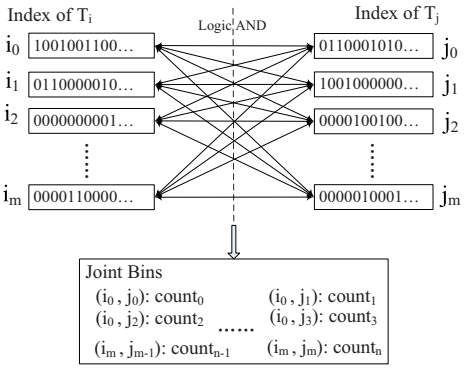


Figure 5: Calculating Conditional Entropy Using Bitmaps

as shown in Equation 3 below. Here, $CFP(j)$ indicates the differences between variable A and variable B when element values are smaller than the value range of the jth bin. The EMD calculation simply sums each $CFP(j)$ together.

$$\begin{aligned} &\text{EMD} = \textstyle\sum_{j=1}^{N} \text{CFP}(j), \\ &\text{CFP}(j) = \text{CFP}(j-1) + Diff(\text{Bin}(A_j), \text{Bin}(B_j)), \\ &\text{CFP}(0) = 0. \end{aligned} \tag{3}$$

Shannon's Entropy: In information theory, the information content of a random variable can be quantified by Shannon's entropy [11]. Constant data (easily predictable) has a low entropy, while apparently random data (uniform probability) has a high entropy. Equation 4 shows the expression to calculate the Shannon's entropy – here, N_A represents the number of distinct values of the attribute A, and the probability distribution functions P_A captures the probability of having each distinct value for A.

$$H(A) = -\textstyle\sum_{j=1}^{N_A} P_A(x_j) \times log_2(P_A(x_j)) \tag{4}$$

Mutual Information: Mutual information is the metric for computing the dependence between two random variables, and shows the amount of shared information between two variables in the number of bits. If the mutual information is low, then the two variables are independent. Conversely, if mutual information is high, one variable provides information about the other. Equation 5 shows the expression to calculate the mutual information. Here, we index the data in the variables or attributes A and B by j and k separately, and use x_j and y_k to represent each distinct value. N_A and N_B represent the number of distinct values of each attribute, and three probability distribution functions, P_A, P_B and P_{AB}, capture the probability of having each distinct value for A, for B, and for a pair of values of A and B, respectively.

$$I(A;B) = \textstyle\sum_{j=1}^{N_A} \sum_{k=1}^{N_B} P_{AB}(x_j, y_k) \times \log(\frac{P_{AB}(x_j, y_k)}{P_A(x_j) P_B(y_k)}) \tag{5}$$

Conditional Entropy: Conditional entropy is defined with the consideration of both self-contained information and the information with respect to others. It is calculated by Shannon's entropy minus mutual information, as shown in Equation 6. The bigger the conditional entropy value of A is, the more information A contains with the respect to B.

$$H(A|B) = H(A) - I(A;B) \tag{6}$$

3.2 Time-steps Selection Using Bitmaps

This subsection describes how to select time-steps more efficiently using bitmaps.

Earth Mover's Distance: Recall that EMD is summed based on *CFP* of each bin, and each *CFP*, in turn, is calculated using the number of elements that are different within each bin pair (bins of two time-steps with the same value range). Hence, the calculation of EMD becomes the calculation of different number of elements for each bin pair between two time-steps. There are two methods to calculate the element differences. The first method is to simply count the number of elements of each bin pair and compute the count differences between them. The second method is to compute the element differences for each bin pair considering spatial information. Because most of the scientific data involves multi-dimensional spaces, calculating elements differences considering their spatial positions is necessary in many cases. For each bin pair, we need to scan each data element inside one bin and find if there is a match at the same position of another bin.

The question we focus on is of computing EMD accurately using only bitmaps, and not the original data. We have developed an algorithm, which we describe now (Figure 4 shows the algorithm pictorially). As a background, first consider the algorithm using full data. First, we have to scan data from both time-steps T_i and T_j to classify the data elements into bins. Then, we have to compare the elements of T_i and T_j inside each bin pair and compute the number of elements that are different. Both steps are time-consuming for large-scale data. In comparison, from Figure 4, we can see that with the help of bitmaps, we already have bitvectors (bins) of both T_i and T_j. For the first bin difference calculation method, we only need to perform 1-bits count operations for each bitvector pair of T_i and T_j. Because each 1-bit inside the bitvector indicates that there is one data element inside current bin, if we choose the same binning scale as when using the full data, we can get the same element differences for each bin pair and hence the same EMD result. For the second calculation method, we only need to perform m number of bitwise *XOR* operations between T_i and T_j to calculate the element differences for each bin pair. Logic *XOR* operation sets the target bit to 1 if the same position of the two inputs has different bit values (one bit is 1 which means there is a data element in current position of one bin, and another bit is 0 which means there is no matching element for the same position of another bin). Hence, we can quickly compute the different number of elements between bin pairs without scanning through each element inside the original data. Also, we are able to generate same element differences for each bin pair considering spatial differences compared with the full data method. This way, we can achieve much better efficiency compared to the full data method without any accuracy loss for the same binning scale.

Conditional Entropy: Now, consider the calculation of conditional entropy, which is the difference between Shannon's entropy and mutual information. Conditional entropy can be computed using the individual value distribution of time-steps T_i and T_j, and the joint value distribution of T_{ij}. Here, the key step is to generate

the joint value distribution. Figure 5 shows the process of calculating conditional entropy using bitmaps. While using the original data, we need to scan through each element of both T_i and T_j at least twice to decide the binning scale and then to generate individual and joint value distributions. Using bitmaps, because each bitvector corresponds to one value or value range, the individual value distributions of T_i and T_j are already generated during the bitmaps generation process. From the figure we can see that, using bitmaps, we only need to perform fast bitwise *AND* operations between m bitvectors of T_i and m bitvectors of T_j to generate all the joint bins, which is efficient. Logic *AND* operation sets the target bit to 1 only if the same position of both inputs contain 1-bit values, which means that each time we count 1 for current joint bin T_{ij} only if both the value of T_i and the value of T_j in current position satisfy the value range of current joint bin. Hence, the joint value distribution generated by bitmaps is the same as that generated by the full data, assuming the same binning scale.

4. OFFLINE ANALYSIS USING BITMAPS

We can use bitmaps (only) to perform different kinds of offline analyses. In our previous work, we have demonstrated that we can perform approximate data aggregation, data spatial join, correlation query, incomplete data analysis and subgroup discovery using bitmaps [2, 30, 38, 39]. In this section, we will focus on correlation mining.

4.1 Motivation for Correlation Mining

There is often a need for complex analyses over datasets generated by scientific simulations. Such analyses can be classified into two categories: individual variable analysis and correlation analysis. Individual variable analysis involves analysis over each variable or attribute independently, and can take the form of data subsetting, data aggregation, data mining or visualization. Much of the existing work, especially in data visualization, has focused on individual variable analysis. However, more recently, several efforts [5, 35] have focused on studying the relationship among multiple variables and making interesting scientific discoveries based on such analysis.

In our previous work [30], we proposed an interactive framework to support correlation queries over multiple variables. Using our tool, users can submit different SQL queries to specify the data subsets (either value-based or dimension-based subsets) they are interested in for correlation analysis. Our tool is able to calculate correlations of the subsets among multi-variables and return the result to users. With the help of bitmaps, the entire process can be executed very efficiently.

However, one open challenge remaining in this work is that the users often do not know what subsets could provide interesting correlation results. Thus, there is a need for a *correlation mining* algorithm, and not just *correlation query processing*.

4.2 Correlation Mining Using Bitmaps

In this work, we focus on correlation mining between two variables. *Mutual Information*, discussed in the previous section, is used here as an indicator of correlations between two variables. Our goal is to find the data subsets (either value or spatial subsets) with a high value of mutual information.

$$I(A;B) = \sum_{j=1}^{N_A} \sum_{k=1}^{N_B} I(A_j;B_k), \quad I(A_j;B_k) \geq 0. \tag{7}$$

$$\begin{array}{l} P_A(x) = P_{A_1}(x) = P_{A_2}(x); \\ P_B(y) = P_{B_1}(y) = P_{B_2}(y); \\ P_{AB}(x,y) = \frac{P_{A_1B_1}(x,y)+P_{A_2B_2}(x,y)}{2},\ P_{A_1B_1} > P_{A_2B_2}; \\ I(A;B) < I(A_1;B_1). \end{array} \tag{8}$$

In our solution, the mining process for value-based subsets and spatial-based subsets follows different orders. If we analyze the expression of mutual information in Subsection 3.1, we can see that mutual information is accumulated based on the probabilities of value subsets. If the mutual information of a bigger value range is low, all its value subsets must contain lower mutual information. As shown in Equation 7, the mutual information $I(A,B)$ is summed by the mutual information of each value subset $I(A_j;B_k)$. Because $I(A_j;B_k)$ is non-negative, we can see that $I(A,B)$ is always larger than or equal to the value of any subset $I(A_j;B_k)$. Hence, we use a top-down method for efficient value-based subsets mining, exploiting multiple levels of bitmaps. However, the situation is different for spatial subsets. We cannot draw the same conclusion if a bigger spatial area contains low mutual information. This is because it is highly possible that one spatial sub-area contains high mutual information while other sub-areas contain low mutual information, which make the final result low. A simple example is shown using Equation 8. Here we divide the data into two sub-spatial areas. To simplify the scenario, we assume these two sub-areas contain the same number of elements and also the same individual value distributions for both variable A and B ($P_A(x) = P_{A_1}(x) = P_{A_2}(x), P_B(y) = P_{B_1}(y) = P_{B_2}(y)$). The difference is that one sub-area has bigger joint value distribution than another ($P_{A_1B_1} > P_{A_2B_2}$). In such case, we can easy see that while the individual value distribution is the same, the joint distribution of entire area P_{AB} value is the average of two sub-areas $P_{A_1B_1}$ and $P_{A_2B_2}$. Because the mutual information is proportional to the joint value distribution, and P_{AB} is smaller than $P_{A_1B_1}$, we can easily find that the mutual information of $I(A;B)$ is smaller than its sub-area $I(A_1;B_1)$. Hence, for spatial subsets, we follow a bottom-up order to calculate mutual information.

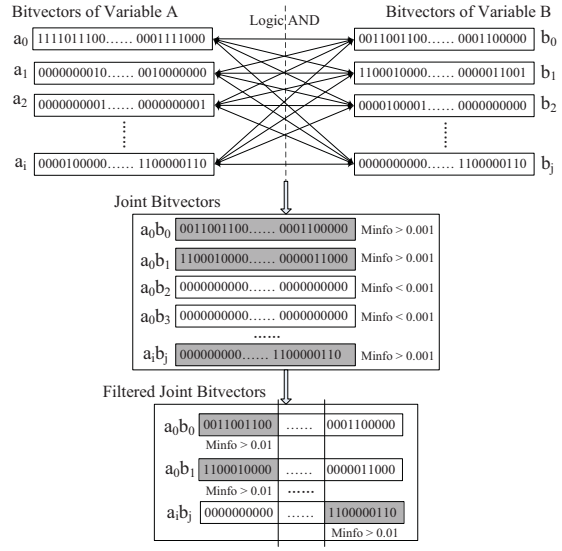


Figure 6: Correlation Mining Using Bitmaps

Without bitmaps, we have to manually divide the entire dataset into a huge number of values and spatial units and then calculate the mutual information between each unit pair, which is extremely time-consuming. With the help of bitmaps, although we follow different orders for value and spatial subsets mining, we do not need to process them separately. Figure 6 shows the process of correlation mining using one-level bitmaps, which comprises three steps. In the first step, we generate joint bitvectors based on logic *AND* operations between bitvectors of A and B. We also count the number of 1-bits for each joint bitvectors during the bitwise *AND* operations. The total number of bitwise operations are $i \times j$, where i is the number of bitvectors of variable A and j is the number of bitvectors of variable B. The second step is a pruning step. Although we generate $i \times j$ number of joint bitvectors, the 1-bits are

distributed sparsely over the joint bitvectors. A pruning step is necessary to filter those joint bitvectors with small correlations. Here we calculate the mutual information for each joint bitvector and filter the joint bitvector if the result is smaller than a threshold T. The rule to decide T here is that even if all the 1-bits of this joint bitvector is located within the same spatial unit, we still consider it as uncorrelated. In such a case, we assume all elements inside this joint bitvector (corresponding to the value subsets of two variables) are uncorrelated and further processing is not required. In the third step, for the remaining joint bitvectors, we partition each bitvector into basic sub-spatial units and generate the 1-bits distribution over all units. We keep only those units with mutual information larger than another threshold T' and filter the others. All the remaining units are considered as spatial subsets with high correlations. Hence, with the help of bitmaps, we only need the bitwise *AND* and *Count* operations to detect all the interesting value or spatial subsets with high correlations.

Algorithm 2: Correlation_Mining($bitVectorsA, bitVectorsB$)

```
1:  for each vectorA in bitVectorsA do
2:    for each vectorB in bitVectorsB do
3:      jointBitVector = LogicAND(vectorA, vectorB);
4:      valueMutualInfo =
        CalMutualInfo(vectorA, vectorB, jointBitVector);
5:      if valueMutualInfo >= THRESHOLD1 then
6:        for unitBegin = 0; unitBegin < dataSize;
          unitBegin+ = unitSize do
7:          spatialMutualInfo =
            CalMutualInfo(jointBitVector, unitBegin, unitSize);
8:          if spatialMutualInfo >= THRESHOLD2 then
9:            AddResult(unitBegin, unitSize, jointBitVector);
10:         end if
11:       end for
12:     end if
13:   end for
14: end for
```

Algorithm 2 shows the pseudo-code of correlation mining between two variables. Line 1 to line 3 iterates each bitvector of variable A and B, perform logic *AND* operation between each bitvector pair and generate joint bitvectors. Line 4 calculates the mutual information for the current joint bitvector. If the current mutual information is larger than a threshold (shown in line 5), we iterate each spatial unit of the joint bitvector and calculate the mutual information within each spatial unit (line 6 to line 11). Finally those spatial units with mutual information larger than another threshold are kept in the final result. All the other values and spatial units are treated as uncorrelated.

There are two optimizations in our work. For multi-dimensional dataset, to keep the multi-dimensional feature of each spatial unit, we use the Z-order curve [27] to iterate over the dataset during the bitmaps generation process. This way, when we partition the joint bitvectors, the basic spatial unit is the size of the smallest unit of Z orders. We can also choose different Z order granularity to match our efficiency goals. Moreover, because usually bitmaps are constructed at multiple levels, the higher level bitmaps contains smaller number of bins with bigger bin ranges, whereas, it is the other way around for the lower level. For efficiency consideration, we begin with high-level bitmaps to quickly filter the low correlated value subsets. Then we only look at the low-level bitvectors belonging to the high-correlated bitvectors of high-level bitmaps.

5. EXPERIMENTAL RESULTS

In this section, we report results from a number of experiments conducted to evaluate our in-situ bitmaps generation and bitmaps based data analysis algorithms. We designed a series of experiments with the following goals: (1) We compare the in-situ analyses time cost and memory consumption between *bitmaps* and the *full data* method, and show that our method can achieve better efficiency and less memory cost for different simulations on different multi-core systems. (2) We compare the efficiency of two cores allocation strategies (*Shared Cores* and *Separate Cores*) and show that a good core allocation strategy can generate bitmaps efficiently. (3) We show that bitmaps can also improve the data analyses efficiency in a parallel in-situ scenario. (4) We show that bitmaps based correlation mining (offline analyses) has much smaller time cost than the *full data* method. (5) We compare the efficiency and accuracy between in-situ sampling and in-situ bitmaps methods and discuss the scenarios where in-situ bitmaps is a better method than in-situ sampling.

We use two simulation programs and one scientific dataset for our experiments. Heat3D simulation [1] is developed to estimate the effect of different geologic structures on heat flow. A 3D mesh can be created to simulate heat flows. The variable generated by the Heat3D application is *temperature*. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (Lulesh) simulation [15] is an application which calculates the motion of materials relative to each other when subject to forces. It approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. It is built on the concept of an unstructured hex mesh, and a node on the mesh is a point where mesh lines (edges) intersect. The variables of each node include *coordinates*, *force*, *velocity vector*, and *acceleration*. Lulesh simulation is a complex simulation with more time and memory cost (to hold mesh lines). The dataset we used for correlation mining is generated by the Parallel Ocean Program (POP) [14], which is an ocean circulation model (the simulation code for POP was unavailable to us). The reason we use this dataset here is it contains multiple variables and some of them have strong correlations within either the value or spatial subsets. POP has a grid resolution of approximately 10 km (horizontally), and vertically it has a grid spacing close to 10 m near the surface, increasing up to 250 m in the deep ocean. The total number of variables in the dataset is 26, and each variable is modeled with either two dimensions (longitude and latitude) or three dimensions (longitude, latitude and depth). The data is stored in the NetCDF format.

For our single node experiments, we use two machines with large number of cores. One machine contains 32 Intel Xeon x5650 CPUs and 1 TB memory, and is located at the Ohio Super-computing Center (OSC). The second machine (Intel MIC) contains 60 Intel Xeon Phi coprocessors and 8 GB memory. Both these machines were chosen because they have a large number of cores on each node, a characteristic that will be common in the near future. In addition, the Intel MIC node has a small amount of memory per core, which is another expected feature for nodes of future supercomputers. For experiments in a distributed memory (cluster) environment, we used 32 nodes with 12 Intel Xeon x5650 CPUs and 48 GB memory from Oakley Cluster of OSC.

5.1 Efficiency and Memory Cost Comparison for In-Situ Analysis

The experiments in this subsection compare the efficiency and memory cost of in-situ analysis between the *full data* method and the *bitmaps* method.

Figures 7 and 8 compare the in-situ analysis efficiency between *full data* and *bitmaps* methods for Heat3D simulation. In our experiment, the simulation generates 100 time-steps and we select 25 time-steps from them using the *Fixed Length Partitioning* method, as described in Section 3.1. The selection metric used here is *conditional entropy*. Figure 7 shows execution on machine with the Intel Xeon and 1 TB memory. The dimension scale of Heat3D is $800 \times 1000 \times 1000$ and the output data size per time-step is 6.4 GB. Figure 8 shows execution on the Intel MIC with 8 GB memory. Because of the memory limitation, the dimension scale here is set as $200 \times 1000 \times 1000$, and the output data size per time-step is 1.6 GB.

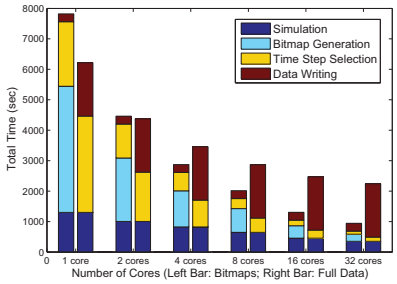


Figure 7: Heat3D: Selecting 25 Time Steps from 100 - Xeon

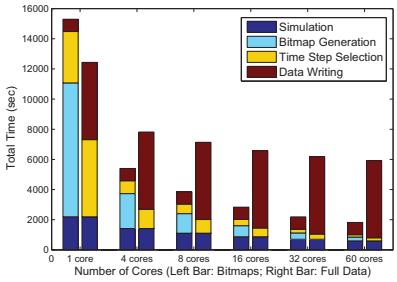


Figure 8: Heat3D: Selecting 25 Time Steps from 100 - MIC

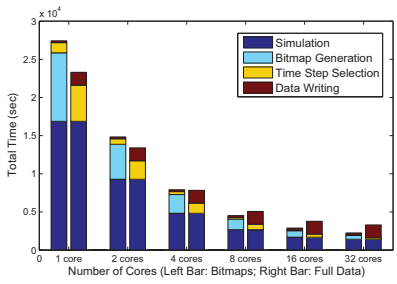


Figure 9: Lulesh: Selecting 25 Time Steps from 100 - Xeon

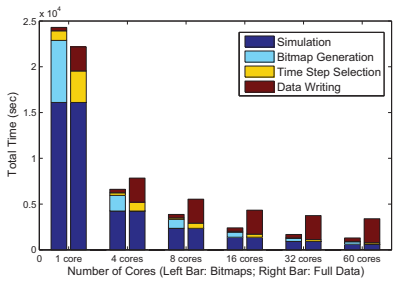


Figure 10: Lulesh: Selecting 25 Time Steps from 100 - MIC

Heat3D simulation has regular temperature changes but the number of distinct values is relatively low. The number of bitvectors (bins) we used ranged from 64 to 206, depending on the temperature range of different time-steps. The binning scale is set to retain 1 digit after the decimal point for each element. The execution time reported for the *bitmaps* method includes the simulation time, the in-situ bitmaps generation time (include both binning time and bitvector compression time), the time-steps selection time using bitmaps, and the time spent outputting bitmaps for the selected time-steps. The execution time reported for the *full data* method includes the simulation time, the time-steps selection time using full data, and the output time for the data from the selected time-steps.

From Figure 7 we can see that the *bitmaps* method has the additional bitmaps generation step, and it has a larger time cost than *full data* method if we only use 1 or 2 cores. However, it greatly reduces the time for time-steps selection and data writing. With the help of bitmaps, we achieved a speedup from 1.38x to 1.50x for time-steps selection (we support fast bitwise operations instead of scanning through 12.8 GB data for each calculation), and achieved a speedup around 6.78x for data writing (the bitmaps size is much smaller than the data size) compared with the *full data* method. The critical observation is that as an increasing number of cores are used, both bitmaps generation time and time-steps selection time is reduced almost linearly, whereas the data output time remains the same. From the figure we can see, the data writing time becomes the major bottleneck after we use 4 cores. If we consider the total execution time, our method can achieve a speedup from 0.79x to 2.37x compared with the *full data* method. More broadly, considering that nodes in current and future HPC machines have large number of cores but limited I/O bandwidth, we can see the clear advantage of bitmaps based approach.

Figure 8 compares the efficiency between the two methods on Intel MIC. Intel MIC contains a larger number of cores for computing, but the I/O bandwidth is even lower. As a larger number of cores are used, we are able to achieve a significant speedup. As a result, our method can achieve a speedup from 0.81x to 3.28x compared with the *full data* method on MIC.

Figures 9 and 10 compare the in-situ analysis efficiency between *full data* and *bitmaps* method for the Lulesh simulation. We select 25 time-steps out of 100 time-steps using the *Fixed Length Partitioning* method, and the selection metric here is *Earth Mover's Distance*. Lulesh is a more complex simulation which generates nodes and edges among nodes. Each node is formed by four variables *Coordinates*, *Force*, *Acceleration* and *Velocity*, and each variable contains data value in three scales *X*, *Y* and *Z*. There are a total of 12 data arrays (each array size is equal to the number of data nodes) for each time-step, and we support in-situ analysis based on all of them. Figure 9 shows execution on the machine with Intel Xeon and 1 TB memory. The total number of nodes generated by Lulesh is 64 MB and the total data size for analysis is 6.14 GB (Edges are used to help calculate nodes value, so we do not include edge data here). Figure 10 shows execution on Intel MIC with 8 GB memory. Because of the memory limitation, the total number of nodes is 8 MB here and the data size for analysis is 768 MB. The number of bitvectors (bins) ranges from 89 to 314, depending upon the number of distinct values in the output at each time-step.

From Figure 9 we can see that, compared with Heat3D simulation, Lulesh is a more complex simulation and has much larger simulation time. For all the cases, the bitmaps generation time and the time-steps selection time is much smaller than the simulation time. Moreover, we are able to achieve a good speedup for time-steps selection using *Earth Mover's Distance*. We only need to perform hundreds of *XOR* operations instead of scanning 12.28 GB data for each calculation. The speedup compared with *full data* method for time-steps selection is from 3.45x to 3.81x. As was the case with Heat3D simulation, our method achieves better efficiency as we use more cores, and the speedup ranges from 0.84x to 1.47x. Figure 10 compares the efficiency between the two methods using Intel MIC. As we had observed in our experiments with the Heat3D simulation, because the MIC system has a larger number of cores and slower disk I/O speed, we are able to achieve even better speedup compared with the *full data* method. The speedup here is from 0.92x to 2.62x.

Figure 11 compares the memory cost between the two methods. In this experiment we kept 10 time-steps in memory for selection. The major memory cost for *Heat3D* using *full data* method is 1 previous selected time-step, 1 intermediate time-step (designed by the simulation) and 10 current time-steps. In comparison, the memory

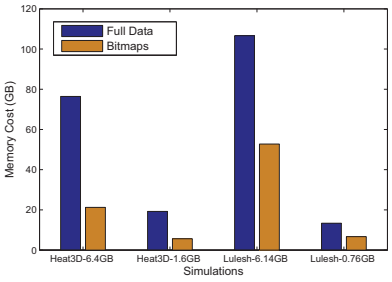


Figure 11: Memory Cost Comparison

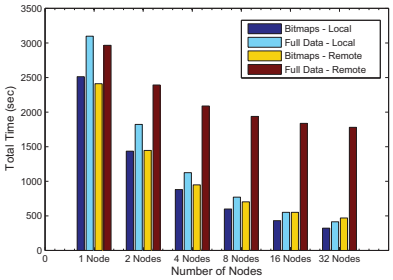


Figure 13: Scalability of Bitmaps (Heat3D on Xeon)

cost using *bitmaps* method is 1 intermediate time-step, 1 current time-step (to simulate next time-step), 1 previous selected bitmap, and 10 current bitmaps. Because bitmaps have much smaller size than the original data, the bitmaps method requires 3.59x (6.4 GB data) and 3.39x (1.6 GB data) smaller memory than the *full data* method. Lulesh is a more complex simulation. Beside the node storage cost, a large amount of memory is used to store the edges, which are used to calculate nodes. However, even with the extra memory cost, our method still has much smaller memory cost, which is 2.02x (6.14 GB) and 1.99x (0.76 GB).

5.2 Comparing Core Allocation Strategies

The experiments compare the simulation and bitmaps generation time between two core allocation strategies, which are the *shared cores* and *separate cores* strategies. Recall that the *shared cores* method involves assigning all the available cores to simulation and bitmaps generation in sequence, whereas the *separate cores* method involves dividing the total cores into two sets and assigning one set to simulation and another to bitmaps generation.

Figure 12 shows the execution times with these two strategies for Heat3D simulation and Lulesh simulation using Intel Xeon and Intel MIC systems. Sub-figure 12(a) compares the total time cost (simulation time plus bitmaps generation time) over 100 time-steps for the Heat3D simulation. The total number of cores used here is 28 (Intel Xeon), and the data size is 6.4 GB. From the figure, we can see that the left bar c_{all} indicates the *shared cores* method. All 28 cores are first used for simulation and then bitmaps generation, and the process is repeated. The right bar group ($c_i\_c_j$) indicates the *separate cores* method. Here *i* is the number of cores used for simulation, and *j* is the number of cores used for bitmaps generation. With the allocation $c_{12}\_c_{16}$, which indicates that we use 12 cores for simulation and 16 cores for bitmaps generation, the average time cost of simulation and bitmaps generation is the closest to each other and hence we are able to achieve the best efficiency. As indicated previously, Heat3D is a very simple numerical program, which also does not scale well with increasing number of cores. Thus, best results are obtained when more cores are dedicated to bitmaps generation than the simulation itself. The time cost of c_{all} is larger than $c_{12}\_c_{16}$. This is because of limited scaling of the simulation program – for example, the speedup is only 1.3x when we use 28 cores compared with 12 cores. This shows that it is not necessary to use all the cores for simulation. When we have a large number of cores, we can use the extra cores for bitmaps generation, which will not have a significant impact on the simulation speed.

Sub-figure 12(b) compares the total execution time using Heat3D simulation on Intel MIC system (using 56 cores). The data size is 1.6 GB. From this figure we can see that when we allocate 32 cores for simulation and 24 cores for bitmaps generation ($c_{32}\_c_{24}$), we can achieve the best efficiency. Again time cost of c_{all} allocation is bigger than $c_{32}\_c_{24}$. Sub-figure 12(c) compares the total time cost using Lulesh simulation on Intel Xeon system. We still use 100 time-steps here and the data size is 6.14 GB. Recall that Lulesh simulation has much bigger simulation time. From the figure we can see that, $c_{20}\_c_8$ achieved the best efficiency, which means we only need to use 8 cores for bitmaps generation. Since most simulation programs involve a large amount of computing, we can expect to see good results by dedicating only a small fraction of the resources for bitmaps generation.

5.3 Efficiency Improvement in Parallel In-Situ Environment

The experiment in this subsection shows that our bitmaps based in-situ analyses method can scale quite well in a cluster setting. Moreover, in clusters or distributed memory machines where data has to be stored to disks that are not local to the node, we show how the bitmaps based method further improves efficiency.

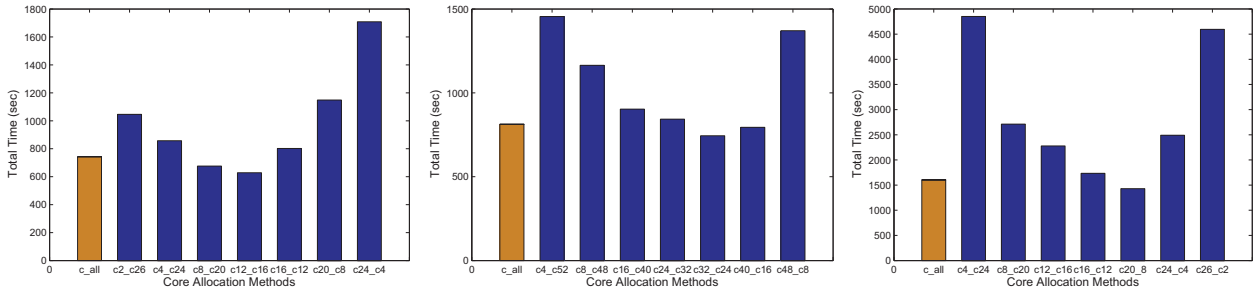
Figure 13 shows the scalability of our method compared with the *full data* method. Same as the previous experiments, the time cost of *full data* method includes the simulation time, the time-steps selection time, and the data I/O time (either writing data to local disk or transferring data to a remote server). The time cost of *bitmaps* method includes the simulation time, the bitmaps generation time, the time-steps selection time, and the data I/O time (writing or transferring bitmaps instead). Here we used 32 computing servers from Oakley Cluster of OSC and 1 remote data server with around 100 MB/sec bandwidth connected to the compute cluster.

The simulation used here is Heat3D, which requires communication (MPI) among machines to update the boundary information. The data size is 6.4 GB. The number of machines varies from 1 to 32, and each machine uses 8 cores for parallel processing. We generate a total of 100 time-steps and select 25 out of them. The selection metric is *Conditional Entropy*. From the figure we can see that, although our method has an extra step for bitmaps generation, we can still achieve better speedup than the *full data* method. Our method saves both time-steps selection and data I/O time cost, and it also scales quite well. The *Bitmap - Local* and *Full Data - Local* methods directly write the data or bitmaps to the local disk of each machine. We can see that our method achieved a 1.24x to 1.29x speedup here. In many cases, the simulated data from multiple computing servers need to be sent to one remote data server for storage, which makes our method achieve further speedup as we have much smaller data transfer size. Using network with around 100 MB/s bandwidth, the *Bitmaps - Remote* method can achieve a 1.24x to 3.79x speedup compared to *Full Data - Remote* method.

5.4 Efficiency Improvement for Offline Analysis

In this experiment, we compares efficiency of correlation mining between *bitmaps* and *full data* method. Here we used the *ocean* dataset generated by the Parallel Ocean Program (POP).

Figure 14 compares the two methods using different data sizes. The two variables we used here are *temperature* and *salinity*. The data size of each variable varies from 1.4 GB to 11.2 GB. From the figure we can see that for all the cases, the *bitmaps* method has smaller execution time than the *full data* method. This is because of a number of reasons. First, our method has much smaller



(a) Heat3D - Xeon (b) Heat3D - MIC (c) Lulesh - Xeon

Figure 12: Efficiency Comparison between Shared Cores and Separate Cores Allocation Strategies

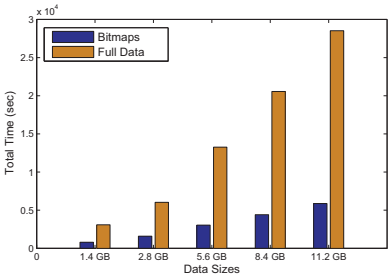


Figure 14: Efficiency of Correlation Mining - POP

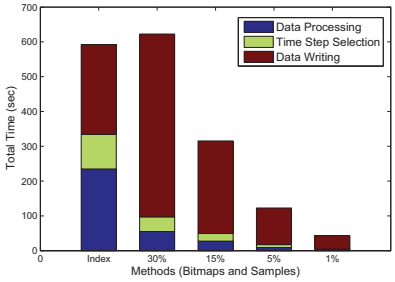


Figure 15: Efficiency Comparison between Sampling and Bitmaps - Heat3D

data loading time and memory cost compared with the *full data* method because bitmaps are much smaller in size. Moreover, the mining process of our method involves performing $m \times n$ bitwise AND operations (*m* is the number of bitvectors of *temperature* and *n* is the number of bitvectors of *salinity*), filtering the uncorrelated units, and followed by 1-bits count operations based on partitioned spatial units. Both logic *AND* operations and 1-bits count operations can be executed very efficiently. On the other hand, the *full data* method needs to reorganize the entire data into small value and spatial units and then do an exhaustive calculation within each unit (The same filter rule is also applied to *full data* method, but it still has much bigger calculation cost). Also multi-level indices can further improve the efficiency, while building different scales of value range based on original data is time consuming. As a result, our method is able to achieve a 3.83x to 4.91x speedup compared with the *full data* method. The larger the dataset size, the better speedup we can achieve. It is also important to note that there is no accuracy loss compared with the *full data* method. This is because both methods use the same binning scale to calculate mutual information. Note that number of bins can be varied with either of the methods, but larger (smaller) number of bins will lead to higher (lower) execution time for both methods.

5.5 Comparing Bitmaps and Sampling

Bitmaps can be viewed as a data reduction method. One simple approach for data reduction is sampling – i.e., simply selecting a smaller number of output elements for further processing. The experiments in this subsection compare the efficiency and accuracy between in-situ sampling and in-situ bitmaps based analyses. The machine we used here are Intel Xeon with 32 cores.

Figure 15 compares the execution time between *bitmaps* and *sampling* method. We used Heat3D simulation here, which selects 25 time-steps out of 100 and the size of each time-step is 6.4 GB. The number of cores used here is 32. The *Bitmaps* method includes simulation, bitmaps generation, time-step selections using bitmaps and outputting bitmaps. The *Sampling* method includes simulation, down-sampling, time-steps selection using sample and sample writing.

The left bar shows the time cost of bitmaps method and the right bar group shows the time cost of sampling method with different sampling levels (from 30% to 1%). From the figure we can see that the sampling method, which only selects a subset of the data from the original dataset, can generate different sample levels efficiently, while bitmaps generation, which requires binning and compression, has much bigger time cost than sampling. Moreover, the data size after sampling is effectively reduces, which also greatly improved the efficiency for the time-steps selection. However, from the figure we can see that the *bitmaps* method still achieves better efficiency than the *sampling* method using 30% samples. This is because the major bottleneck here is still the disk I/O speed. With the help of multiple cores within the node, we can significantly reduce the time differences for data processing (bitmaps generation or sampling) and time-steps selection between these two methods.

Besides the time cost, another big advantage of bitmaps method is that, using bitmaps for time-steps selection and correlation calculation does not incur any accuracy loss. In comparison, sampling involves loss of accuracy. Figure 16 shows the accuracy loss with different sampling levels, when sampling is performed before time-steps selection. For bitmaps method, the accuracy is the same as using the original dataset if we choose the same binning scales. For sampling method, we calculate the conditional entropy values between each time-step pairs and then compute the absolute value differences between the sample result and the original result. Then, we use a *Cumulative Frequency Plot* (CFP) to represent the absolute conditional entropy differences for all pairs. In CFP, a point *(x,y)* indicates that the fraction y of all calculated relative value differences are less than x. Because the value differences should be as small as possible, it implies that a method with the curve to the left has a better accuracy than the method with the curve to

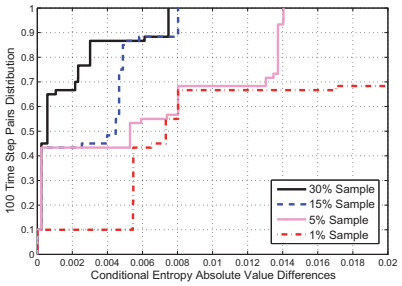


Figure 16: Accuracy Lost for Time Step Selection - Heat3D

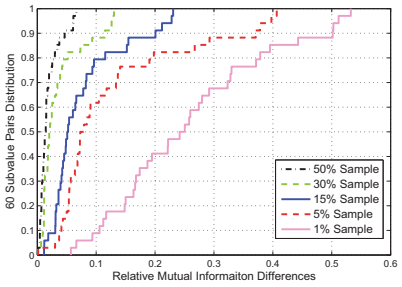


Figure 17: Accuracy Lost for Correlation Mining - POP

the right. From the figure, we can see that the smaller the sample level we use, the more accuracy loss we incur, which is not surprising. If we calculate the relative accuracy loss using the expression $(original\_result - sample\_result)/original\_result$ for each pair, using 30%, 15%, and 5% sample size incurs an average of 21.03%, 37.56%, and 58.37% information loss, respectively. Hence, we can see that although the sampling method can achieve better efficiency than our method if we use a small sample level (less than 30%), the information loss is very significant. On the other hand, 30% of the sample does not help the performance, but still involves a measurable loss of accuracy.

Figure 17 shows the accuracy loss with different sampling levels for offline analyses (Correlation Mining). For correlation mining, we continued to use POP data. The variables used here are *temperature* and *salinity* with 1.4 GB data size for each variable. Here we first divided the variables into 60 spatial and value subsets. Then we calculate the *mutual information* between these two variables within each subset. The value differences between the original dataset and samples are also shown in CFP plot. From the figure, we can also see that while bitmaps method does not have any information loss, the smaller sample level we use, the more accuracy loss we incur. Compared with the original result, using 50%, 30%, 15%, and 5% sample size incurs an average of 3.14%, 7.56%, 10.15%, and 17.03% accuracy loss, respectively

6. RELATED WORK

In this section, we will first describe the existing in-situ analysis work and then discuss pertinent work on correlation analysis.

In-situ analysis frameworks have been developed by several groups in recent years. ADIOS [17, 23] makes it possible to independently select the I/O methods for each grouping of data in an application (based on both I/O patterns and the underlying hardware). GoldRush [45], which is built over ADIOS, uses fine-grained scheduling methods to harvest the otherwise idle resources to run in-situ data analytics efficiently. Several other efforts [12, 25] focus on improving I/O efficiency by identifying and leveraging I/O access patterns. GLEAN [34] is a flexible and extensible framework which takes application, analysis, and system characteristics into account to facilitate simulation-time data analysis and I/O acceleration. The DIY [28] is an in-situ analyses prototype library of scalable core components for the decomposition and movement of data. The Damaris/Viz [10] studied different core allocation strategies (*time partitioning* vs *space partitioning*) with the consideration of memory constraints for in-situ data visualization, while our work is more focused on bitmap generation.

A recent effort [16] has demonstrated that the bitmap index generation efficiency can be greatly improved in an in-situ parallel environment. In their work, they analyzed the index comression and IO time and demonstrated that bitmap index compression efficiency can be greatly improved using multi-core. As we stated earlier, our contributions are in developing strategies for core allocation, and use of bitmaps as a data profile for various analytics steps. A potentially different in-situ compression method is presented in another recent paper [20]. Specifically, the DIRAQ framework provides a parallel *in-situ*, in network data encoding and reorganization technique that enables the transformation of simulation output into a query-efficient form. To the best of our knowledge, the method will not allow steps such as entropy or mutual information calculation or correlation analysis.

Analysis of multiple variables and their relationships in scientific simulation outputs has been an ongoing topic of research. In a very recent work, Biswas *et al.* [5] presented an information theoretic framework for exploring multivariate datasets in a *top-down* manner, where they divide the variables into groups based on their information overlap, and then identify representative variables from each group to conduct further relationship analysis. Wang *et al.* [35] used information theory for exploring the causal relationship among the variables of a time-varying multivariate dataset. In an earlier work, Jänicke *et al.* [13] applied the dimensionality reduction on the high dimensional data where each dimension was analogous to a variable in the multivariate context. Another well-known multivariate exploration technique was developed by Di Yang *et al.* [44]. They use a Nugget Management System (NMS), where the nuggets represent the information that the users are interested in. Several authors have surveyed existing multivariate data analysis techniques [9, 40]. Almost none of these efforts have focused on in-situ analyses and scalability limitations, especially, the challenges associated with a memory-constraint platform.

Our recent work demonstrated how bitmaps can be used to accelerate computation of correlations, in both parallel and distributed setting [30]. In comparison, this work makes several contributions. First, we have demonstrated the feasibility (and even benefits of) bitmap generating in in-situ settings. Second, we have presented novel algorithms for using bitmaps for time-step selection (which includes steps like computation of entropy and Earth Mover Distance). Next, we have used develolped a novel algorithm for correlation mining using bitmaps. Finally, we have experimentally demonstrated how the use of bitmaps for summarization can be benefitial in architectures that are memory and I/O constraint.

7. CONCLUSIONS

Dealing with I/O and memory constraints is rapidly becoming the most critical challenge for data analytics associated with scientific data. In this paper, we have proposed an approach, which uses bitmaps as a compact representation of data that can be used for various in-situ and offline analyses steps. The key algorithmic contributions of the paper include a method for generating in-situ bitmaps with different core allocation strategies, methods for computing correlation metrics and Earth Mover's Distance (EMD) using only bitmaps, and a correlation mining algorithm using bitmaps. Because binning is used for these steps even when the original data is used, use of bitmaps does not lead to any loss of accuracy, whereas memory, I/O, and computational costs are reduced.

Our future efforts will focus on extending this work in at least two directions. First, we will like to evaluate this approach on other architectures like GPUs. Second, we will consider other in-situ and offline analytics tasks for acceleration using bitmaps.

Acknowledgements

This work was supported by DOE Office of Science, Advanced Scientific Computing Research, under award number DE-DC0012495, program manager Lucy Nowell, and by NSF under the award ACI-1339757.

8. REFERENCES

[1] Heat3d simulation. http://http://dournac.org/info/parallel_heat3d.

[2] Sameh Abdulah, Yu Su, and Gagan Agrawal. Accelerating data mining on incomplete datasets by bitmaps-based missing value imputation. In *Proceedings of the 7th International Conference on Advances in Databases, Knowledge, and Data Applications*, 2015.

[3] James Ahrens, Sébastien Jourdain, Patrick OÂŠLeary, John Patchett, D Rogers, and M Peterson. An imagebased approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014.

[4] G. Antoshenkov. Byte-aligned bitmap compression. In *Data Compression Conference, 1995. DCC'95. Proceedings*, page 476. IEEE, 1995.

[5] Ayan Biswas, Soumya Dutta, Han-Wei Shen, and Jonathan Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, 2013.

[6] Udeepta D Bordoloi and H-W Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494. IEEE, 2005.

[7] J. Chou, K. Wu, O. Rübel, M.H.J.Q. Prabhat, B. Austin, E.W. Bethel, R.D. Ryne, and A. Shoshani. Parallel index and query for large scale data analysis. In *SC*, 2011.

[8] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[9] M.C.F. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: a survey. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):378–394, 2003.

[10] Matthieu Dorier, Roberto Sisneros, Tom Peterka, Gabriel Antoniu, and Dave Semeraro. A nonintrusive, adaptable and user-friendly in situ visualization framework. In *Proceedings of the third international symposium on Large-Scale Data Analysis and Visualization*. IEEE, 2013.

[11] Stefan Gumhold. Maximum entropy light source placement. In *Visualization, 2002. VIS 2002. IEEE*, pages 275–282. IEEE, 2002.

[12] Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, and Xian-He Sun. I/o acceleration with pattern detection. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 25–36. ACM, 2013.

[13] H. Jänicke, M. Bottinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1459–1466, 2008.

[14] PW Jones, PH Worley, Y. Yoshida, JB White III, and J. Levesque. Practical performance portability in the parallel ocean program (pop). *Concurrency and Computation: Practice and Experience*, 17(10):1317–1327, 2005.

[15] Ian Karlin, Jeff Keasler, and Rob Neely. Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, August 2013.

[16] Jinoh Kim, Hasan Abbasi, Luis Chacon, Ciprian Docan, Scott Klasky, Qing Liu, Norbert Podhorszki, Arie Shoshani, and Kesheng Wu. Parallel in situ indexing for data-intensive computing. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 65–72. IEEE, 2011.

[17] Scott Klasky, Hasan Abbasi, Jeremy Logan, Manish Parashar, Karsten Schwan, Arie Shoshani, Matthew Wolf, Sean Ahern, Ilkay Altintas, Wes Bethel, et al. In situ data processing for extreme-scale computing. In *Proc. Conf. Scientific Discovery through Advanced Computing Program (SciDAC.11)*, 2011.

[18] Peter M Kogge and Timothy J Dysart. Using the top500 to trace and project technology and architecture trends. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 28. ACM, 2011.

[19] Peter M. Kogge and John Shalf. Exascale computing trends: Adjusting to the "new normal"' for computer architecture. *Computing in Science and Engineering*, 15(6):16–26, 2013.

[20] Sriram Lakshminarasimhan, David A Boyuka, Saurabh V Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E Papka, and Nagiza F Samatova. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd international symposium on HPDC*, pages 1–12. ACM, 2013.

[21] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jacqueline Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031. IEEE Press, 2014.

[22] Teng-Yok Lee and Han-Wei Shen. Visualization and exploration of temporal trend relationships in multivariate time-varying data. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1359–1366, 2009.

[23] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[24] Robert McLay, Doug James, Si Liu, John Cazes, and William Barth. A user-friendly approach for tuning parallel file operations. In *Proceedings of the international conference on SC*, pages 229–236. IEEE, 2014.

[25] James Oly and Daniel A Reed. Markov model prediction of i/o requests for scientific applications. In *Proceedings of the 16th international conference on Supercomputing*, pages 147–155. ACM, 2002.

[26] P. O'Neil and D. Quass. Improved query performance with variant indexes. In *ACM Sigmod Record*, volume 26, pages 38–49. ACM, 1997.

[27] V. Pascucci and R.J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing, ACM/IEEE 2001 Conference*, pages 45–45. IEEE, 2001.

[28] Tom Peterka, Robert Ross, Attila Gyulassy, Valerio Pascucci, Wesley Kendall, Han-Wei Shen, Teng-Yok Lee, and Abon Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 105–112. IEEE, 2011.

[29] Yu Su, Gagan Agrawal, and Jonathan Woodring. Indexing and parallel query processing support for visualizing climate datasets. In *Proceedings of the 41th IEEE/ACM International Conference on Parallel Processing*, pages 249–258. IEEE, 2012.

[30] Yu Su, Gagan Agrawal, Jonathan Woodring, Ayan Biswas, and Han-Wei Shen. Supporting correlation analysis on scientific datasets in parallel and distributed settings. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 191–202. ACM, 2014.

[31] Xin Tong, Teng-Yok Lee, and Han-Wei Shen. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 49–56. IEEE, 2012.

[32] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modelling. In *Computer Graphics Forum*, volume 22, pages 689–700. Wiley Online Library, 2003.

[33] Ivan Viola, Miquel Feixas, Mateu Sbert, and Meister Eduard Groller. Importance-driven focus of attention. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):933–940, 2006.

[34] Venkatram Vishwanath, Mark Hereld, and Michael E Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14. IEEE, 2011.

[35] Chaoli Wang, Hongfeng Yu, Ray W Grout, Kwan-Liu Ma, and Jacqueline H Chen. Analyzing information transfer in time-varying multivariate data. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 99–106. IEEE, 2011.

[36] Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Importance-driven time-varying data visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1547–1554, 2008.

[37] Jim Jing-Yan Wang, Xiaolei Wang, and Xin Gao. Non-negative matrix factorization by maximizing correntropy for cancer clustering. *BMC bioinformatics*, 14(1):107, 2013.

[38] Yi Wang, Yu Su, and Gagan Agrawal. A novel approach for approximate aggregations over arrays. In *Submission to the 18th International Conference on Extending Database Technology*, 2015.

[39] Yi Wang, Yu Su, Gagan Agrawal, and Tantan Liu. Scisd: Novel subgroup discovery over scientific datasets using bitmap indices. In *Proceedings of Ohio State CSE Technical Report*, 2015.

[40] Pak Chung Wong and R. Daniel Bergeron. 30 years of multidimensional multivariate visualization. In *Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 3–33, Washington, DC, USA, 1997. IEEE Computer Society.

[41] K. Wu, E.J. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 99–108. IEEE, 2002.

[42] K. Wu, K. Stockinger, and A. Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Scientific and Statistical Database Management*, pages 348–365. Springer, 2008.

[43] Kesheng Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *15th International Conference on Scientific and Statistical Database Management, 2003*, pages 65– 74. IEEE, July 2003.

[44] Di Yang, E.A. Rundensteiner, and M.O. Ward. Analysis guided visual exploration of multivariate data. In *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*, pages 83–90, 2007.

[45] Fang Zheng, Hongfeng Yu, Can Hantas, Matthew Wolf, Greg Eisenhauer, Karsten Schwan, Hasan Abbasi, and Scott Klasky. Goldrush: resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2013.