Declustering two-dimensional datasets over MEMS-based Storage

Hailing Yu Divyakant Agrawal Amr El Abbadi
University of California at Santa Barbara
Computer Science Department
Santa Barbara, 93106
USA
{hailing,agrawal,amr}@cs.ucsb.edu

Abstract

Due to the large difference between seek time and transfer time in current disk technology, it is advantageous to perform large I/O using a single sequential access rather than multiple small random I/O accesses. However, prior optimal cost and data placement approaches for processing range queries over two-dimensional datasets do not consider this property. In particular, these techniques do not consider the issue of sequential data placement when multiple I/O blocks need to be retrieved from a single device. In this paper, we reevaluate the optimal cost of range queries, and prove that, in general, it is impossible to achieve the new optimal cost. This is because disks cannot facilitate two-dimensional sequential access which is required by the new optimal cost. Fortunately, MEMS-based storage is being developed to reduce I/O cost. We first show that the two-dimensional sequential access requirement can not be satisfied by simply modeling MEMS-based storage as conventional disks. Then we propose a new placement scheme that exploits the physical properties of MEMS-based storage to solve this problem. Our theoretical analysis and experimental results show that the new scheme achieves almost optimal results.

1 Introduction

Multi-dimensional datasets have received a lot of attention due to their wide applications, such as relational databases, images, GIS, and spatial databases. Range queries are an important class of queries for multi-dimensional data applications. In recent years, the size of the datasets have been rapidly growing, hence, fast retrieval of relevant data is the key to improve query performance. A common method of browsing geographic applications is to display a low resolution map of some area on a screen. A user may specify a rectangular bounding box over the map, and request the retrieval of the higher resolution map of the specified region. In general, the amount of data associated with different regions may be quite large, and may involve a large number of I/O transfers. However, due to advances in processor and semi-conductor technologies, the gap in access cost between main memory and disk is constantly increasing. Thus, I/O cost will dominate the range query cost. To alleviate this problem, two-dimensional datasets are often uniformly divided into tiles. The tiles are then declustered over multiple disks to improve query performance through parallel I/O. In [14], it is shown that strictly optimal solutions for range queries exist in only a small number of cases. Given a query that retrieves A tiles, the optimal cost is $\lceil A/M \rceil$, where M is the number of I/O devices. A data assignment scheme is said to be strictly optimal if it satisfies the optimal cost. Because it is impossible to find the optimal solution for the general case, several assignment schemes have been proposed [8, 9, 15, 5, 6, 22, 10] to achieve near optimal solutions. However, all prior research, including the optimal cost formulation itself, are based on the assumption that the retrieval of each tile takes constant time and is retrieved independently (random I/O involving both seek and transfer time). This condition does not hold when considering current disk technology. Recent trends in hard disk development favor reading large chunks of consecutive disk blocks (sequential access). Thus, the solutions for allocating two-dimensional data across multiple disk devices should also account for this factor. In this paper, we reevaluate the optimal cost for current disk devices, and show that it is still impossible to achieve the new optimal cost for the general case.

Even though the performance gap between main memory and disks is mitigated by a variety of techniques, it is still bounded by the access time of hard disks. The mechanical positioning system in disks limits the access time improvements to only about 7% per year. Therefore, this performance gap can only be overcome by inventing new storage technology. Due to advances in nano-technology, Micro-ElectroMechanical Systems (MEMS) based storage systems are appearing on the horizon as an alternative to disks [16, 1, 2]. MEMS are devices that have microscopic moving parts made by using techniques similar to those used in semiconductor manufacturing. As a result, MEMS devices can be produced and manufactured at a very low cost. Preliminary studies [18] have shown that stand-alone MEMS based storage devices reduce I/O stall time by 4 to 74 times over disks and improve the overall application run times by 1.9X to 4.4X.

MEMS-based storage devices have very different characteristics from disk devices. When compared to disk devices, head movement in MEMS-based storage devices can be achieved in both X and Y dimensions, and multiple probe tips (over thousands) can be activated concurrently to access data. In [12], Griffin et al. consider the problem of integrating MEMS-based storage into computers by modeling them as conventional disks. In [13], we proposed a new data placement scheme for relational data on MEMS based storage by taking its characteristics into account. The performance analysis showed that it is beneficial to use the MEMS storage in a novel way (similar work has been proposed in [19]). Range queries over two-dimensional data (map or images) are another data-intensive application, thus, in this paper, we explore data placement schemes for two-dimensional data over MEMS-based storage to facilitate efficient processing of range queries. In our new scheme, we tile two-dimensional datasets according to the physical properties of I/O devices. In all previous work, the dataset is divided into tiles along each dimension uniformly. Through the performance study, we show that tiling the dataset based on the properties of devices can significantly improve the performance without any adverse impact on users, since users only care about query performance, and they do not need to know how or where data is stored. The significant improvement in performance further demonstrates the great potential that novel storage devices, like MEMS, have for database applications.

The rest of the paper is organized as follows. Section 2 revisits the optimal cost of a range query of a two-dimensional dataset. In Section 3, we first give a brief introduction of MEMS-based storage; then we adapt the existing data placement schemes proposed for disks to MEMS-based storage. A novel scheme is proposed in Section 4. Section 5 presents the experimental results. We conclude the paper in Section 6.

2 Background

In this section, we first address the problem of existing optimal cost evaluation for range queries over two-dimensional data, then we derive the optimal cost for the current disk devices.

2.1 Existing problem

A disk is organized as a sequence of identically-sized disk pages. In general, the I/O performance of queries is expressed in terms of the number of pages accessed, assuming that the access cost is the same for each page. Each access cost is composed of three elements: *seek time, rotational delay, and transfer time*. We refer to the sum of seek time and rotational delay as *access time*. Based on this assumption, the optimal cost of a query is derived as

$$\lceil A/M \rceil \times (t_{access} + t_{transfer}). \tag{1}$$

Where A is the number of tiles intersecting the query, M is the number of disk devices, t_{access} is the average access time of one page, $t_{transfer}$ is the average transfer time of one page (all the terms have the same meaning throughout the paper). We will refer to equation (1) as the *prior optimal cost*. Because each tile is retrieved by an independent random I/O, in order to improve the access cost of a query, existing data placement schemes decluster the dataset into tiles and assign neighboring tiles to different devices, thus allowing concurrent access of multiple pages.

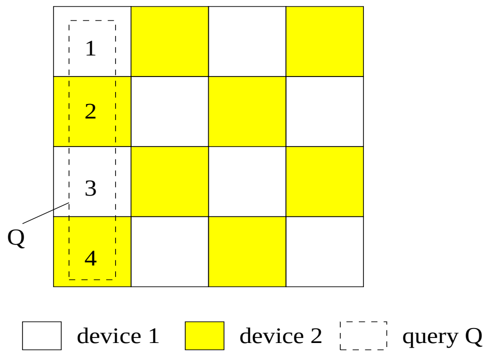


Figure 1: An image picture is placed on two disk devices

In recent years, disk external transfer rates have improved significantly from 4MB/s to over 500MB/s; internal transfer rate is up to 160MB/s. However, access time has only improved by a factor of 2. For a disk with 8KB page size, 160MB/s internal transfer rate, and 5ms access time, the ratio of access time over transfer time is about 100. Thus, current disk technology is heavily skewed towards sequential access instead of random access due to the high cost of access time. This principle can benefit range queries over two-dimensional data, if we could place the related tiles of a query that are allocated on a single disk device sequentially. For example, Figure 1 shows an image with sixteen disk pages (or tiles). Assume that we place this image on two disk devices, according to [14], tiles will be placed as shown in Figure 1. If tile 1 and 3 are sequentially placed on disk device 1 and tile 2 and 4 are placed on device 2, the access cost of the query Q (given by the dashed rectangle in Figure 1) is $1 \times t_{access} + 2 \times t_{transfer}$, instead of $2 \times (t_{access} + t_{transfer})$. Based on this observation, we reevaluate the optimal cost of range queries for hard disks.

2.2 Optimal Cost

Due to the fact that current hard disk is more efficient when accessing data sequentially rather than random access, the advantage of sequential access must be considered while determining the I/O cost of a query, in order to facilitate the fast retrieval of the related tiles in a range query. Ideally, if all tiles related to a query, which are allocated to same device, are placed sequentially, then only one access time is incurred for the query. Thus the optimal retrieval cost is given by the following formula (referred to as *new optimal cost*).

$$t_{access} + \lceil A/M \rceil \times t_{transfer} + (\lceil A/N_t \rceil - 1) \times t_{trackswitch}. \qquad (2)$$

where N_t is the average number of blocks in a track, $t_{trackswitch}$ is the time of switching from a track to one of its neighboring tracks. The first term in the formula is the cost of one access time, because all the related tiles on one device are placed consecutively, and all initial accesses to different disks are performed concurrently. The second term is the maximal transfer time of all devices, which is derived by considering parallelism of multiple disk devices. The third term represents the track switching time if the number of involved tiles $\lceil A/M \rceil$ is too large to be stored on one track. However, it is impossible to achieve the optimal cost for all queries, as shown by the following argument.

In general, the two-dimensional data is divided into $N_1 \times N_2$ tiles. The number of disk devices is M. Consider a query Q_1 with A tiles. Without loss of generality, assume A is much smaller than N_t (thus there is no track switching time), and A is equal to M as shown in Figure 2 (a). Based on the new optimal cost formula (2), each tile in this query is allocated to a different device. Thus, the access cost of this query is

$$t_{access} + \lceil A/M \rceil \times t_{transfer} = t_{access} + t_{transfer}.$$

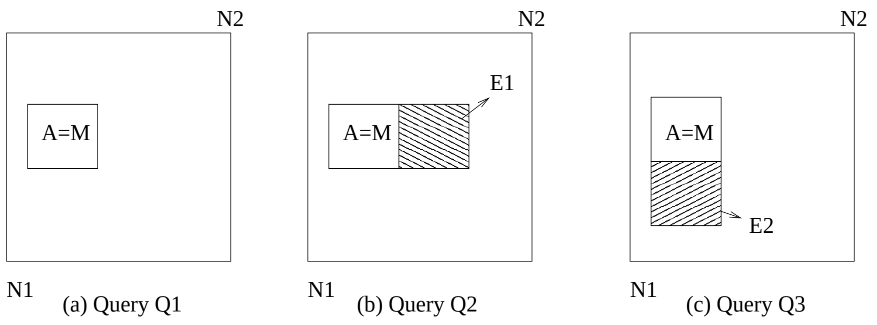


Figure 2: Queries in a $N_1 \times N_2$ data set

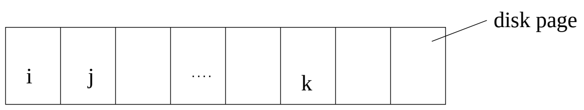


Figure 3: Allocate tiles to the sequential disk pages

This cost is optimal. Figure 2 (b) shows query Q_2 which extends query Q_1 in the X dimension. The number of tiles in query Q_2 is $A + E_1$, and $E_1 = M$. In order to achieve the optimal cost, each device is allocated two tiles, and these two tiles need to be placed sequentially. Alternatively, we extend query Q_1 in the Y dimension to generate a new query Q_3, as shown in Figure 2 (c). Query Q_3 accesses $A + E_2$ tiles, where E_2 also equals M. Therefore, each device needs to place two tiles (one from A and one from E_2) sequentially to achieve the optimal cost. However, it is impossible to achieve the optimal cost for both query Q_2 and Q_3.

Assume tiles i, j, k are on device d, tile i intersects Q_1, tiles i, j intersect Q_2, and tiles i, k intersect Q_3. In order to achieve sequential access for query Q_2, tiles i and j need to be placed next to each other. Thus, tile k can not be placed next to i, as shown in Figure 3. Alternatively, if k is placed next to i, then j can not be placed next to i. Thus, in general, it is impossible to achieve the optimal cost for all queries of a data set, given the physical characteristics of disks.

3 Adapting Disk Allocation Scheme to MEMS Storage

MEMS storage devices are novel experimental systems that can be used as storage. Their two-dimensional structure and inherent parallel retrieval capability hold the promise of resolving the problems disks face during the retrieval of two-dimensional datasets. In this section, we first introduce MEMS-based storage briefly, then we show how to adapt existing disk allocation schemes to MEMS-based storage.

3.1 MEMS-based Storage

MEMS are extremely small mechanical structures on the order of 10μm to 1000μm, which are formed by the integration of mechanical elements, actuators, electronics, and sensors. These micro-structures are fabricated on the surface of silicon wafers by using photolithographic processes similar to those employed in manufacturing standard semiconductor devices. MEMS-based storage systems use MEMS for the positioning of read/write heads. Therefore, MEMS-based storage systems can be manufactured at a very low cost. Several research centers are developing real MEMS-based storage devices, such as IBM millipede [16], Hewlett-Packard Laboratories ARS [2], and Carnegie Mellon University CHIPS [1].

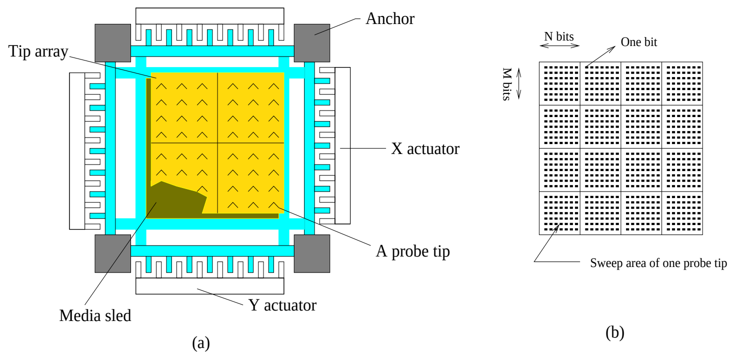


Figure 4: The architecture of CMU CHIPS

Due to the difficulty of manufacturing efficient and reliable rotating parts in silicon, MEMS-based storage devices do not make use of rotating platters. All of the designs use a large-scale MEMS array, thousands of read/write heads (called *probe tips*), and a recording media surface. The probe tips are mounted on micro-cantilevers embedded in a semiconductor wafer and arranged in a rectangular fashion. The recording media is mounted on another rectangular silicon wafer (called the *media sled*) that uses different techniques [7, 16, 1] for recording data. The IBM Millipede uses special polymers for storing and retrieving information [16]. The CMU CHIPS project adopts the same techniques as data recording on magnetic surfaces [1]. To access data under this model, the media sled is moved from its current position to a specific position defined by (x, y) coordinates. After the "seek" is performed, the media sled moves in the Y direction while the probe tips access the data. The data can be accessed sequentially in the Y dimension. The design is shown in Figure 4 (a).

Table 1: Parameters of MEMS-based storage devices from CMU CHIPS

Number of regions	6400 (80×80)
Number of tips	6400
Max number of active tips	1280
Tip sector size	8 bytes
Servo overhead	10 bits per tip sector
Bits per tip region ($M \times N$)	2000×2000
Per-tip data rate	0.7Mbit/s
X axis settle time	0.125ms
Average turnaround time	0.06ms

The media sled is organized into rectangular regions at the lower level. Each of these rectangular regions contains $M \times N$ bits and is accessible by one tip, as shown in Figure 4 (b). Bits within a region are grouped into vertical 90-bit columns called *tip sectors*. Each tip sector contains 8 data bytes, which is the smallest accessible unit of data in MEMS-based storage [7]. In this paper, we will use the CMU CHIPS model to motivate two-dimensional data allocation techniques for MEMS-based storage. The parameters are given in Table 1. Because of heat-dissipation constraints, in this model, not all tips can be activated simultaneously even though it is feasible theoretically. In the CMU CHIPS model, the maximal number of tips that can be activated concurrently is limited to 1280 (which provides the intra-device parallelism). Each rectangular region stores 2000×2000 bits.

3.2 Adapting Disk Placement Approaches

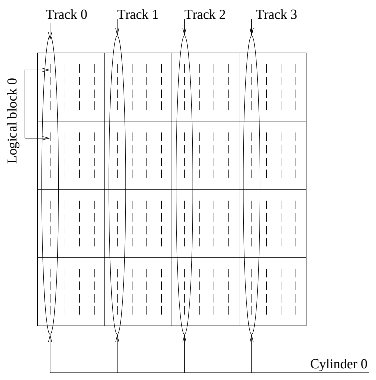


Figure 5: Disk Terminologies for MEMS-based storage

Even though MEMS-based storage has very different characteristics from disk devices, in order to simplify the process of integrating it into computing systems, some prior approaches have been proposed to map MEMS-based storage into disk-like devices. Figure 5 shows this mapping on a MEMS-based storage device with 16 probe tips by using the CMU model [12]. Using disk terminology, a *cylinder* is defined as the set of all bits with identical x offset within a region; i.e., a cylinder consists of all bits accessible by all tips when the sled moves only in the Y direction. For example, Cylinder 0 is shown in Figure 5 and is highlighted by four ellipses. Due to power and heat considerations, only a subset can be activated simultaneously. Thus cylinders are divided into tracks. A *track* consists of all bits within a cylinder that can be accessed by a group of concurrently active tips. In Figure 5, four out of sixteen tips can be activated concurrently, and each cylinder contains 4 tracks. Track 0 to Track 3 of Cylinder 0 are shown in Figure 5. Each track is composed of multiple tip sectors, which contain less data than sectors of disks. Sectors can be grouped into logical blocks. In [12], each logical block is 512 bytes and is striped across 64 tips. Therefore, by using the MEMS-based storage devices in Table 1, there are up to 20 logical blocks that can be accessed concurrently (1280 concurrent tips$/64$ tips $= 20$). Data is accessed based on its logical block number on the MEMS-based storage.

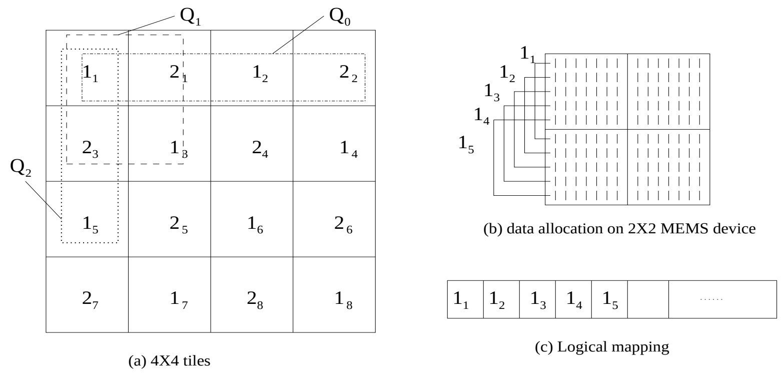


Figure 6: An example of placing tiles on MEMS-based storage

After this mapping, a MEMS-based storage device can be treated as a regular disk. Thus, all prior approaches proposed for disk devices can be adapted to MEMS-based storage, such as Disk Modulo (DM) [8], Generalized Fibonacci (GFIB) scheme [15], Golden Ratio Sequences (GRS) [6], Almost optimal access [5], and other approaches based on replication [22, 10]. All of these approaches are based on the prior optimal cost (1) and do not factor the sequential access property of current disks. Hence, when adapting them to MEMS-based storage devices, this problem still remains unsolved. For example, a two-dimensional dataset with 4×4 tiles is shown in Figure 6 (a). Two MEMS-based storage devices are used to store the data ($M = 2$). The MEMS devices in this example have four probe tips each, and only two of them can be activated concurrently. Based on the prior optimal cost (1), the tiles should be distributed evenly and alternately to the two devices, as shown in Figure 6 (a). The number d_i in each tile means that tile is allocated to MEMS device d, and i stands for the relative position of that tile on the device. In this example, we use two tip sectors to contain the data of one tile (The size of one tile is determined by the maximum size of one data retrieval. In this example, it is $2 \times 8 = 16$). As a result, an allocation is showed in Figure 6 (b) based on the above mapping, Tiles 1_1 to 1_5 can be placed sequentially. Their logical positions are given in Figure 6 (c). This allocation is optimal according to the prior optimal cost (1). In fact, for query Q_0 in Figure 6 (a), this allocation is also optimal according to the new optimal cost (2). However, for Queries Q_1 and Q_2 shown in Figure 6 (a), both of them can not achieve the new optimal solution. Q_1 needs to retrieve the tiles 1_1 and 1_3. Based on the relative layout given in Figure 6 (c), they are not placed sequentially. A similar analysis can be achieved for query Q_2. In fact, using this mapping, MEMS-based storage devices lose some of their characteristics. The intra-device parallelism is not considered at all in this mapping. In [13], we showed that by taking these properties into account, MEMS-based storage can achieve extra performance gain in relational data placement. In the next section, we propose a new scheme for placing two-dimensional data on MEMS-based storage, which exploits the physical properties of MEMS-based storage to achieve sequential-like performance for queries such as Q_1 and Q_2.

4 Parallel MEMS Storage Allocation

In this section, we propose a new data placement scheme based on the physical characteristics of MEMS storage.

4.1 Motivation and Challenges

In Section 2, we argued that it is impossible to achieve the new optimal cost for disk devices. In order to achieve the optimal cost, tiles have to be placed in both X and Y dimensions sequentially, which cannot be realized by one-dimensional storage devices (disk-like devices). Thus, in our new scheme, we need to place the dataset in such a way that tiles can be sequentially accessed in both dimensions (we refer to this as the *two-dimensional sequential access requirement*). Even though MEMS-based storage is arranged in a two-dimensional manner, data can only be accessed in the Y dimension sequentially (which gives sequential access in the Y dimension). According to the design of MEMS-based storage, any movement in the X dimension will result in a new seek. Therefore, we have to solve the two-dimensional sequential access requirement by using other properties of MEMS-based storage.

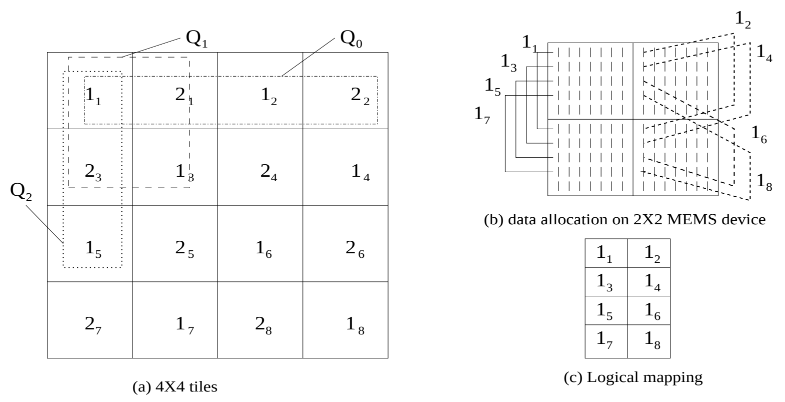


Figure 7: An example by taking the intra-device parallelism into account

Due to power and heat constraints, only a subset of the probe tips can be activated concurrently. Even though tip sectors with the same coordinates cannot all be accessed simultaneously, they can be accessed "almost" sequentially. For example, we show a new data layout for the example in Figure 6. Instead of placing tiles 1_1 and 1_2 along the Y dimension sequentially, we place them over a group of tip sectors with the same coordinates. In order to answer query Q_0, which is shown in Figure 7 (a), we first activate the tip sectors for tile 1_1 (only two out of four tips can be activated concurrently), then turn the media sled around and activate the tip sectors for tile 1_2 without moving in the X dimension. In another sense, query Q_0 is accessed sequentially (no extra seek cost). Based on this principle, we place the tiles shown in Figure 7 (a) over the MEMS storage devices as shown in Figure 7 (b). Its corresponding logical layout is given in Figure 7 (c). Thus, query Q_1 can also achieve an optimal answer ($t_{access} + 2 * t_{transfer}$). However, the access cost of query Q_2 is $2 * t_{access} + 2 * t_{transfer}$ (the track switching time is zero), which is still not optimal (tiles 1_1 and 1_5 are not sequentially allocated, hence it results in an extra access penalty).

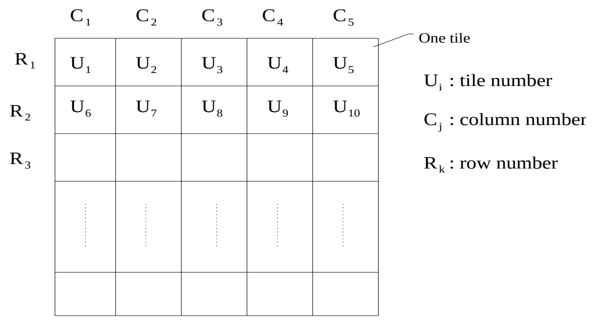
In order to solve this problem and take advantage of sequential access, we could allocate the tiles sequentially along the Y dimension over a single device. The 4×4 dataset shown in Figure 6 and 7 can be allocated in a way shown in Figure 8. This allocation can achieve optimal performance for both Q_0 and Q_1. Furthermore, since the tiles $1_1, 1_3, 1_5$ are placed sequentially, the access cost of query Q_2 will be reduced to $t_{access} + 3 * t_{transfer}$. Even though the new cost is reduced compared to the layout in Figure 7, the data intersecting with query Q_2 is not distributed over the M given devices uniformly (the optimal cost should be $t_{access} + 2 * t_{transfer}$). Based on this discussion, we observe that sequential access and parallelism over multiple devices are conflicting goals. To address this contradiction, in our placement scheme, we first tile the dataset based on the physical properties (rather than tiling the dataset without considering the properties of devices), then we allocate the data in each tile on the M devices evenly to guarantee that each device participates almost equally in transfer time. Note that from a user's point of view, the underlying tiling is not of interest and has no effect on the user's interaction with the dataset; efficiency and fast retrieval is the user's goal.

1_1	2_1	1_2	2_2
1_3	2_3	1_4	2_4
1_5	2_5	1_6	2_6
1_7	2_7	1_8	2_8

Figure 8: Placing the tiles sequentially over a single device

4.2 Tiling the data

In general, two-dimensional data is divided into tiles along each dimension uniformly, and the size of each tile corresponds to one disk page. In our scheme, we divide the data into tiles based on the physical properties of the underlying storage devices, in particular, the number of given devices, M, and their parameters. In Table 1, 1280 of 6400 probe tips can be activated concurrently. Because the smallest accessible unit is a tip sector with 8 bytes, 1280×8 bytes can be accessed concurrently. There are M devices, thus the total size of data which can be accessed simultaneously is $M \times 1280 \times 8$ bytes. This determines the size of a tile. In other words, a tile corresponds to the maximum number of bytes that can be accessed concurrently. After



C_1 C_2 C_3 C_4 C_5
One tile
R_1 R_2 R_3
U_1 U_2 U_3 U_4 U_5
U_6 U_7 U_8 U_9 U_{10}
U_i : tile number
C_j : column number
R_k : row number

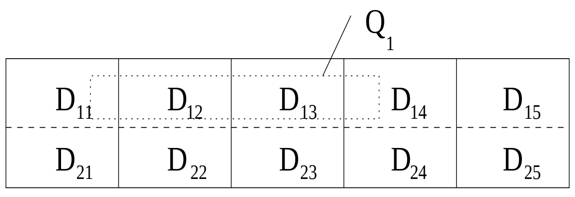
Figure 9: Dividing the two-dimensional data into tiles

determining the size of a tile, the next step is to decide on the number of tiles along each dimension. In our scheme, in order to take advantage of potential concurrent access of tip sectors with the same coordinates, data is divided into $6400/1280$ tiles along the X dimension (i.e., 5 columns), as shown in Figure 10. All the tip sectors allocated to the 5 tiles of a single row on one device have the same coordinates, thus if the total number of tip sectors on one device in a query, which only intersects with the 5 tiles of a single row, does not exceed the maximal number of concurrent tips, they could be retrieved by one transfer. Hence given a dataset of dimensions $W \times L = S$bytes, we divide W into 5 columns. Each tile will have dimensions $\frac{W}{5} \times \frac{M \times 1280 \times 8}{W/5}$. The limitation on the width of the dataset can be addressed by increasing the number of devices, slicing the tile as described in Section 4.3. After determining the number of tiles in the X dimension, the number of tiles along the Y dimension can be computed by the following formula:

$$\lceil S/\text{the data size in each row}\rceil = \lceil S/(M \times 6400 \times 8)\rceil.$$

Where S stands for the total size of the dataset.

4.3 Slicing a Tile



Q_1
D_{11} D_{12} D_{13} D_{14} D_{15}
D_{21} D_{22} D_{23} D_{24} D_{25}

D_{ij}: i stands for the device number;
j is the column number.

Figure 10: An example of placing each tile over M devices

After having divided the data into tiles, we introduce how to distribute the data in each tile over M MEMS-based storage devices. In order to avoid the problem of sacrificing transfer time to achieve sequential access, we distribute the data in each tile over all M devices uniformly. However, the placement has to guarantee that the transfer time of a query on each device is the same. For example, consider a single row of a dataset with 5 tiles, and two MEMS storage devices. For the sake of argument, let us assume the top 1280 eight-byte units of each tile are allocated to device 1 and the bottom 1280 eight-byte units are allocated to device 2, as shown in Figure 10. For a query Q_1 given by the dotted rectangle in Figure 10, it is impossible to achieve optimal transfer time. Even though the data of $D_{11}, D_{12}, D_{13}, D_{14}, D_{15}$ are placed on tip sectors with the same coordinates which have the potential to be accessed concurrently, if the number of tip sectors accessed by Q_1 is larger than the maximal number of concurrent tips, lets say two times larger, then answering Q_1 will take two transfer time. However, in this case, only device 1 is used. If device 2 could retrieve half of the requested data, only one transfer time is necessary (which is the optimal cost). Thus the data in each tile should be distributed over the M devices evenly at the level of tip sectors.

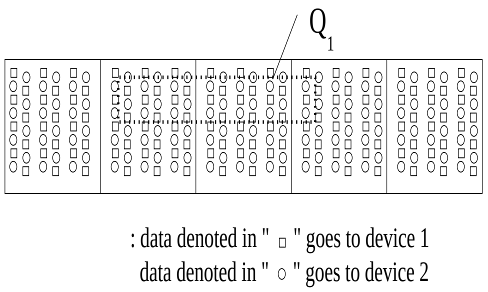

Q_1
: data denoted in " □ " goes to device 1
data denoted in " ○ " goes to device 2

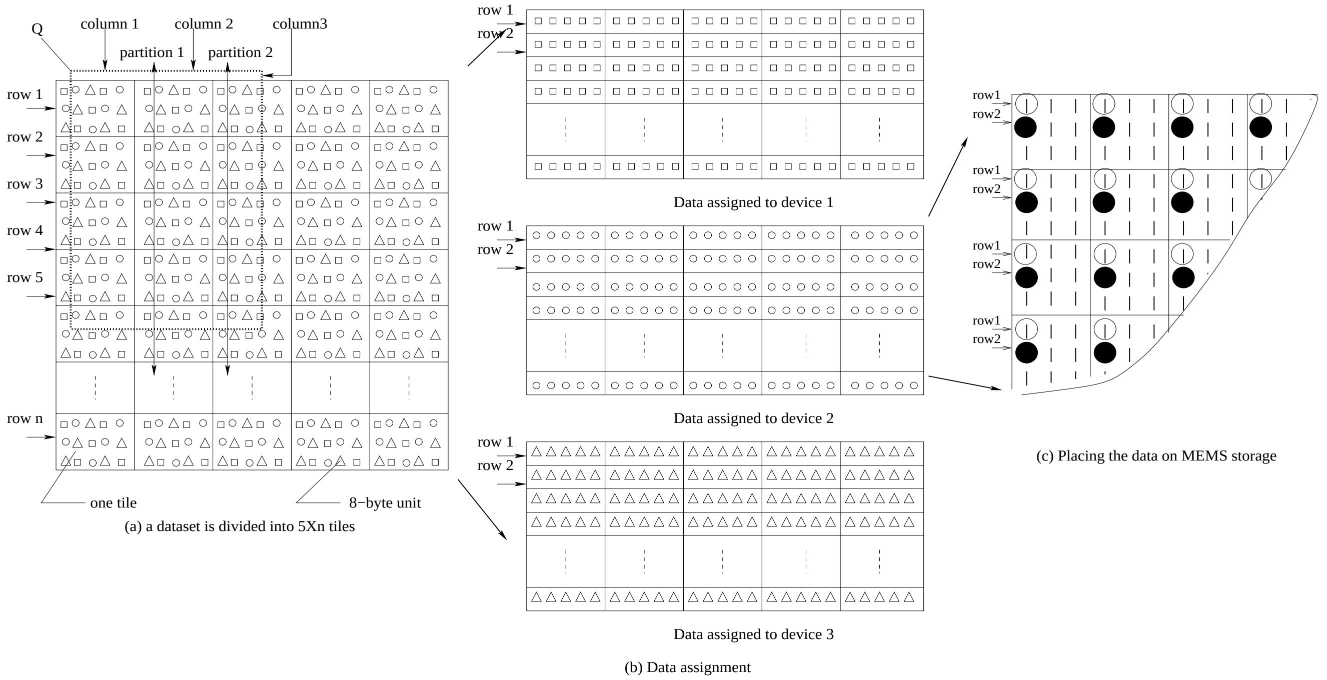
Figure 11: An example of slicing one tile over two devices

In each tile, the data is organized into 8-byte units which correspond to the size of each tip sector. The data allocation at this level needs to distribute the 8-byte units over the M devices evenly. Lets take the dataset shown in Figure 10 as an example. In each tile, the data are organized into 8-byte units, which are highlighted by little squares and circles in Figure 11. If we allocate all 8-byte units highlighted by squares to device 1, and all 8-byte units shown by circles to device 2, as shown in Figure 11, then the involved data is distributed over the two devices evenly. As a result, query Q_1 can be answered optimally. Slicing each tile over multiple devices is similar to declustering a two-dimensional dataset over multiple disk devices, if we treat a tile as a new two-dimensional dataset and each 8-byte unit as a new tile of the dataset. Thus, all the data placement schemes proposed for two-dimensional dataset could be easily adopted. For example, in [5], the authors use a coloring scheme. It is proved that any range query over this dataset is guaranteed to access no more than $\lceil N_u/M \rceil + \gamma$ 8-byte units where $\gamma = O(logM)$ and N_u is the total number of 8-byte units in a tile. We adopt this scheme for allocating data in one tile on multiple MEMS devices.

4.4 Algorithms

In this section, we describe the procedures for placing the data on MEMS storage devices, and given a query, how to retrieve the relevant data.

Based on the parameters given in Table 1, the number of tip sectors of a single device which are allocated to one tile is the maximal number of concurrent tips (1280). There are 6400 tips in total, as a result, all the tip sectors with the same coordinates can form 5 ($6400/1280$) tiles of maximal concurrent tip sectors. All the tip sectors in these five tiles have the potential to be accessed simultaneously. Furthermore, any $M \times 1280 \times 8$ bytes in a row have the potential to be accessed concurrently. The layout among different tiles and across different rows is considered in a similar way to the one used in [13]. The generic data allocation procedure



Q
column 1
column 2
column3
partition 1
partition 2
row 1
row 2
row 3
row 4
row 5
row n
one tile
8−byte unit
(a) a dataset is divided into 5Xn tiles
row 1
row 2
Data assigned to device 1
Data assigned to device 2
Data assigned to device 3
(b) Data assignment
row1
row2
(c) Placing the data on MEMS storage

Figure 12: Placing the tiles of a dataset over M MEMS devices

is given in Algorithm 1. We use an example to explain this algorithm. In Figure 12 (a), given three MEMS storage devices ($M = 3$), a dataset is divided into a $5 \times n$ tile grid. In each tile, the little circles, squares, and triangular stand for 8-byte units, that will be allocated to different devices, as shown in Figure 12 (b). The 8-byte units on row1 are mapped to row1 on each device in Figure 12 (b). Figure 12 (c) shows how to allocate the data assigned to each device on the physical storage device. Assume we start placing the data at the coordinates (x, y), then the data of row1 is placed on the tip sectors with coordinates (x, y), which are highlighted by circles in Figure 12 (c). Based on the data provided in Table 1, in order to decrease the seek time, we first consider the allocation of the tip sectors which have the same x coordinate as row1, and y coordinate differs by one from row1. As a result, row2 is placed on the tip sectors with coordinates $(x, y+1)$, which are highlighted by filled circles in Figure 12 (c). This process continues until the y coordinate reaches the boundary, when the tip sectors are allocated in the next column (coordinate x differs by one from the last row).

In general, the dataset is placed at two levels: the lower level places each tile over the M devices, while the upper level places the rows within a single device. Based on this layout, we now describe how to answer a query (data retrieval). Given a query Q, assume Q intersects rows i, i+1,...,i+k in continuous order. The data retrieval procedure is given in Algorithm 2. In this algorithm, there are two cases to consider:

- Case 1: For any single device[1], the number of tip sectors in any row, which intersects the query, is not larger than the maximal number of concurrent tips.

  The media sled is moved to the coordinates (x_i, y_i) of the row i, then the tips which are in charge of the

[1]The data in each tile can only be distributed over the M devices almost evenly.

Algorithm 1 Data Placement

```
1: The maximum number of concurrent tips: $N_{concurrent}$;
2: The total number of tips: $N_{total}$;
3: The number of tiles in a row: $N_{row} = \lceil N_{total}/N_{concurrent} \rceil$;
4: The number of given devices: $M$;
5: The size of each tip sector: $S_{tipsector}$;
6: The size of one tile: $S_{tile} = N_{concurrent} \times M \times S_{tipsector}$;
7: The size of each row of tiles: $S_{row} = N_{total} \times M \times S_{tipsector}$;
8: The size of the dataset: $S_{dataset}$;
9: The number of tiles in a column: $N_{column} = S_{dataset}/S_{row}$;
10: Organizing the data in each tile into $S_{tipsector}$-byte units;
11: Using any disk allocation scheme to distribute the data in each tile over $M$ devices
12: for each device do
13:   Move the media sled to the initial position with coordinates $(x, y)$;
14:   $moveDirection = down$;
15:   $i = 0$;
16:   while $i \leq N_{row}$ do
17:     Allocate tip sectors with coordinates $(x, y)$ to data tiles at row $i$;
18:     if $moveDirection = down$ then
19:       $y = y + 1$;
20:       if $y$ is out of range of movement of the media sled then
21:         $y = y - 1, x = x + 1$;
22:         Move the media sled to the position $(x, y)$, and turn it around;
23:         $moveDirection = up$
24:       end if
25:     else
26:       $y = y - 1$;
27:       if $y$ is out of range of movement of the media sled then
28:         $y = y + 1, x = x + 1$;
29:         Move the media sled to the position $(x, y)$, and turn it around;
30:         $moveDirection = down$
31:       end if
32:     end if
33:     $i = i + 1$;
34:   end while
35: end for
```

Algorithm 2 Data Retrieval

```
1: Given a query $Q$;
2: $Q$ intersects the dataset by rows $i, i+1, ..., i+k$ in continuous order;
3: The maximum number of concurrent tips: $N_{concurrent}$;
4: for each device do
5:    The number of involved tips in a row: $N_{involved}$;
6:    Move the media sled to the coordinates of tip sectors for row $i$: $(x_{start}, y_{start})$;
7:    $x = x_{start}, y = y_{start}$;
8:    $moveDirection = down$;
9:    $retrievedrows = 0$;
10:   if $N_{involved} \leq N_{concurrent}$ then
11:      while $retrievedrows \leq k$ do
12:         Activate all involved tips, and retrieve the data;
13:         if $moveDirection = down$ then
14:            $y = y + 1$;
15:            if $y$ is out of range of movement of the media sled then
16:               $y = y - 1, x = x + 1$;
17:               Move the media sled to the position $(x, y)$, and turn it around;
18:               $moveDirection = up$
19:            end if
20:         else
21:            $y = y - 1$;
22:            if $y$ is out of range of movement of the media sled then
23:               $y = y + 1, x = x + 1$;
24:               Move the media sled to the position $(x, y)$, and turn it around;
25:               $moveDirection = down$
26:            end if
27:         end if
28:         $retrievedrow += 1$;
29:      end while
30:   else
31:      The number of concurrent groups: $N_{group} = \lceil N_{involved}/N_{concurrent} \rceil$;
32:      For each of these groups
33:      Turn around the media sled
34:      if $moveDirection = down$ then
35:         $moveDirection = up$;
36:      else
37:         $moveDirection = down$;
38:      end if
39:      Use the same procedure as the one when $N_{involved} \leq N_{concurrent}$;
40:   end if
41: end for
```

intersected tip sectors are activated, and the corresponding data with these coordinates are retrieved. Then the media sled moves to the coordinates of the next row, $i + 1$, to access the data. Based on Algorithm 1, we know that any two neighboring rows differ either in their x coordinates by one, or y coordinates by one, which means, data is either sequentially retrieved in the Y dimension or at most involves one track switching time (move from one column to its neighboring column). On the other hand, there is no unnecessary data (which is not included in the range of Q) retrieved.

- Case 2: For some devices, the number of tip sectors in any row involved in Q is larger than the maximal number of concurrent tips.

  In this case, the relevant data in each row can not be retrieved simultaneously. For any single device, the intersected tip sectors of each of row i, i+1,...,i+k are organized into concurrent groups[2] (The number of tip sectors in the final group may be less than the maximal number of concurrent tips). For example, Figure 12 (a) gives a query Q by a dotted rectangle. The partitions 1 and 2 regroup the query range into three new columns, as shown in Figure 12 (a), these three new columns with the original rows, divide the query range into new tiles (the new tiles in column 3 contain less data than the other two new tiles). Based on the tile allocation scheme, the 8-byte units in the new tiles are distributed over the M devices evenly. Thus, we can adopt the same retrieval procedure as in Case 1 to access the data in each new column. After finishing access of one new column, instead of moving the media sled back to the coordinates of row1 every time (it will introduce a new seek), the media sled is just turned around and changes its moving direction, then the tips in another concurrent tip group are activated to access the data (tip sectors can be accessed in both directions of the Y dimension).

Based on the whole procedure of our data placement scheme, from tiling the dataset to query answering, we can see that the new scheme considers the parallelism of a single MEMS-based storage device, thus, the maximal number of concurrent tips are used to retrieve the relevant data. The sequential access property is considered in the row-wise manner. This solves the problem, which exists on disks, that the necessary data are sequentially accessed without retrieving unrelated data to a query.

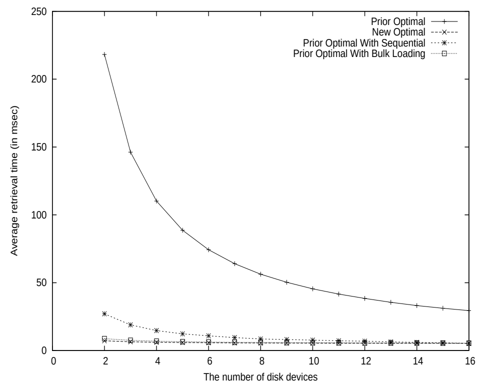
5 Experimental Results

In this section, we first compare the new optimal cost (2) with the prior optimal one (1) through experiments on disks, and show that the prior data placement schemes can not achieve the new optimal cost in general. Then we evaluate our new data placement scheme for MEMS-based storage devices.

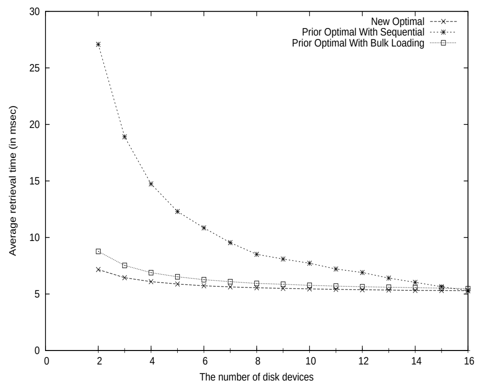
5.1 Evaluating the Optimal Cost on Disks

In this section, we experimentally explore the relationship between the prior and new optimal costs (Equation (1) and (2)). We also investigate ways in which I/O scheduling may try to take advantage of sequential I/O on disks to improve the performance of algorithms designed based on the prior optimal cost. We do not evaluate any specific existing data placement schemes, because all of them are based on the prior optimal cost, they can not outperform the prior optimal cost. We therefore assume the data placement schemes can achieve the prior optimal cost. However, these schemes based on the prior optimal cost do not discuss how

[2]The tip sectors in each row have the same coordinates, thus they have the potential to be accessed simultaneously.



(a)



(b)

Figure 13: Performance evaluation of a dataset with $N_1 = N_2 = 16$ tiles

data is placed on the individual disks. We therefore consider two cases, one when data is placed randomly on each disk and the other where data is allocated sequentially on the disks. Finally, we also investigate the performance when the disk access mechanism retrieves large chunks of data even though the chunks may contain unnecessary data in the middle. This is based on the fact that seek time is much more expensive than transfer time. Thus, the transfer time is sacrificed to improve seek time [20, 21, 17]. We refer to this approach as *bulk loading*. Hence we consider three cases.

- Random access: Tiles assigned to the same disk are placed randomly. In this case, the access of different tiles is an independent random access.
- Sequential access: Tiles allocated to a single device are placed row by row. Thus, the two-dimensional tiles are mapped onto one-dimensional I/O devices (refer to Figure 6 as an example). The tiles intersecting with queries are not retrieved sequentially if they are not placed consecutively. In this case, no unnecessary tiles are accessed (or transfered).
- Bulk loading: The data layout is the same as the one in sequential access. However, we use bulk loading, thus some unnecessary tiles may be retrieved.

Table 2: Parameters of a disk drive

Transfer time	$0.05ms$
Access time	$5ms$
The average number of blocks on a track	300
Track switching time	$2.5ms$

The comparison is based on the average cost of all possible queries [15]. We first average the cost of all queries with the same size(i.e., the same number of tiles), then we take the average of all possible queries with different sizes. In our experiments, for the sake of simplicity, we place the first tile from the beginning of a track. We use the parameters listed in Table 2 [4, 3].

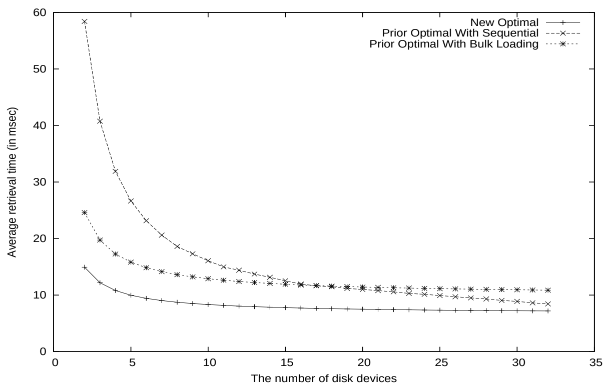


Figure 14: Performance evaluation of a dataset with $N_1 = N_2 = 32$ tiles

The first experiment is conducted on a dataset with 16×16 tiles, which can be contained on one track, thus, there is no track switching time. The result is shown in Figure 13. Figure 13 (a) shows the curves of the average retrieval time over different numbers of disk devices. The prior optimal cost is much larger than the other three curves, thus we do not consider it in our later experiments. The corresponding curves without the prior optimal shown in Figure 13 (a) are given in Figure 13 (b) to highlight the I/O gap that exists. Bulk loading access pattern outperforms sequential access. Figure 14 shows the results of a dataset with 32×32 tiles. The tendency of the three curves is similar to Figure 13 (b). However, the difference between the new optimal cost and the other two is larger than in Figure 13 (b), which is due to the track switching time. The prior optimal cost (1) does not consider the relative positions of involved tiles. In other words, sequential access and bulk loading involves more track switching time than the optimal cost. As the number of devices is increased, sequential access outperforms bulk loading, because sequential access has less track switching time than bulk loading.

5.2 Evaluating the New Placement Scheme on MEMS

In this section, we evaluate our new placement scheme for MEMS-based storage devices. We use the coloring scheme proposed in [5] to distribute the data in each tile. We compare our new scheme with the new optimal cost. We also evaluate the prior optimal layout for MEMS-based storage with sequential access and bulk loading. In these cases, we use the CHIPS disk-based modeling of MEMS storage described in Section 3.2. The seek time and transfer time we use for the MEMS-based storage devices are $1.46ms$ and $0.129ms$ respectively [11]. The turn-around time is $0.06ms$ which is much smaller than seek time and transfer time. The number of tip sectors in the Y dimension of a region is 22 [13]. Because we use a different standard to tile two-dimensional dataset, in our comparison, we transfer the ranges of queries in the original data tiles to our new layout. Thus, we compare the same set of range queries for different schemes.

Figure 15 shows the performance when the number of tiles in a query is varied over 20×20 dataset and 4 MEMS storage devices. The average retrieval cost is obtained by taking the average access cost of all possible ranges with the same number of tiles. Our new placement scheme always outperforms the sequential access, and outperforms bulk loading in most queries. The little difference between our new scheme and the optimal cost is due to the turn-around time. Because the dataset is composed of 20×20 tiles, each tile contains $8K$ bytes. According to our tiling scheme, the dataset is reorganized into 5×20 tiles,

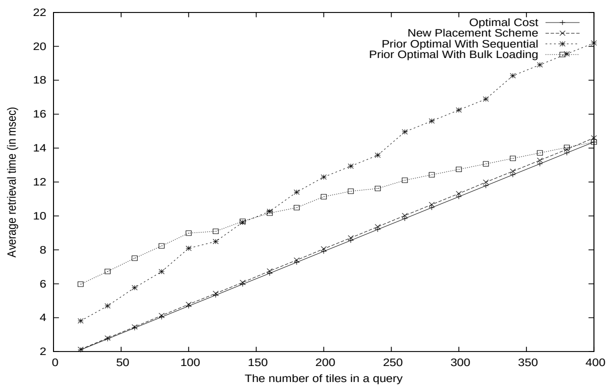


Figure 15: Performance evaluation of a dataset with original $N_1 = N_2 = 20$ tiles over four MEMS-based storage devices

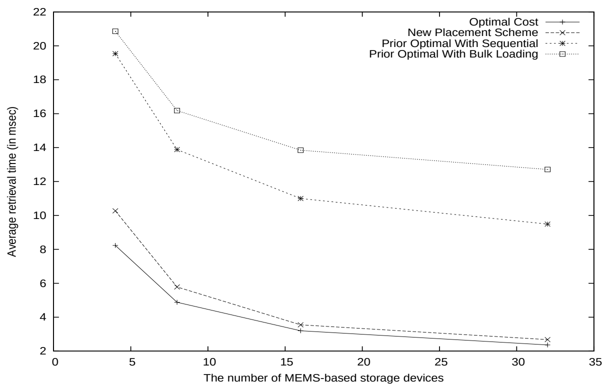


Figure 16: Performance evaluation of a dataset with original $N_1 = N_2 = 80$ tiles

each of the tile in the new scheme contains $M \times 1280 \times 8 = 4 \times 1280 \times 8$ bytes. For the sake of simplicity, we place the rows at the beginning of regions, i.e., row1 has coordinates $(0, 0)$. In each region, there are 22 tip sectors in the Y dimension, thus, there is no track switching time involved. In the new scheme, the media sled needs to move forwards and backwards to retrieve the tiles in different columns. When the number of tiles in a query is very large (almost the entire dataset), bulk loading slightly outperforms the new scheme. When a large number of tiles are involved in a query, then the number of unnecessary tiles retrieved in bulk loading is very small. However, in the new scheme, there is the extra cost of turn-around time. When the number of tiles is small, sequential access has better performance than bulk loading, because bulk loading retrieves too many unrelated tiles, as shown in Figure 15.

In order to further evaluate the average performance of these four methods, we compared their average retrieval cost for all sizes of queries by varying the number of MEMS-based storage devices and using a dataset with 80×80 tiles. Our new scheme performs better than both sequential access and bulk loading, as shown in Figure 16. The difference between our scheme and the optimal cost is due to the turn-around time and the column-switching time. In our scheme, the column-switching time is larger than for the optimal cost, since column-switching time in optimal cost is not affected by the relative positions of the accessed

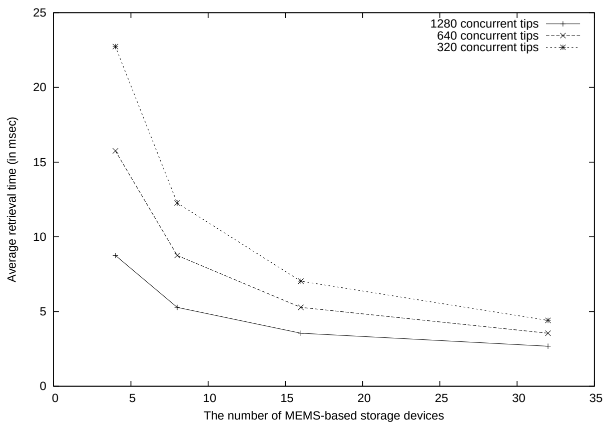


Figure 17: Performance evaluation of a dataset with original $N_1 = N_2 = 80$ tiles

tiles.

Due to power and heat dissipation constraints, only a subset of the tip sectors can be activated concurrently. In our experiment, we used CMU MEMS-based storage: 1280 of total 6400 tips can be activated simultaneously. In order to study the effect of this constraint on our new scheme, we vary the number of concurrent tips (1280, 640, 320), the results are shown in Figure 17. With the increase in the number of MEMS storage devices, all these curves have similar trends. The time difference among them is due to the transfer time and the turn around time. For example, a 1280-concurrent-tips device has twice the bandwidth of a 640-concurrent-tips device, i.e., a 1280-concurrent-tips device can transfer data faster than a 640-concurrent-tips devices. Based on our scheme, the dataset is organized into 10 columns for 640-concurrent-tips devices compared to 5 columns for 1280-concurrent-tips devices, thus 640-concurrent-tips devices will turn the media sled more often than 1280-concurrent-tips devices. Hence, 640-concurrent-tips devices incur more turn-around time than 1280-concurrent-tips devices.

6 Conclusion

The cost of range queries over two-dimensional datasets is dominated by disk I/O. Declustering and parallel I/O techniques are used to improve the query performance. Since the optimal cost can not be realized in general, several approaches have been proposed to achieve near optimal cost. However, the physical properties of current disks are not considered. In this paper, we reevaluate the optimal query cost for current disks, which favor sequential accesses over random accesses. We also demonstrate that it is impossible to achieve the new optimal cost on disks in general, since the new optimal cost of range queries requires two-dimensional sequential access. This is not feasible in the context of disks, since disks are single-dimensional devices. In this paper, we propose to use MEMS-based storage to overcome this inherent problem in disks. We develop a new data placement scheme for MEMS-based storage that takes advantage of its physical properties, where the two-dimensional sequential access requirement is satisfied. The new scheme allows for reorganization of query ranges into new tiles which contain more relevant data, thus we can maximize the number of concurrent active tips to access the relevant data. In the new scheme, we also relax the traditional tiling approach (which has never been specified explicitly) by tiling datasets based on the physical properties of MEMS-based storage. Our theoretical analysis and experimental results show that our new scheme can achieve almost optimal performance.

References

[1] CMU CHIP project. 2003. http://www.lcs.ece.cmu.edu/research/MEMS.

[2] Hewlett-packard laboratories atomic resolution storage, 2003. http://www.hpl.hp.com/research/storage.html.

[3] Reference guide - hard disk drives. 2003. http://www.storagereview.com/guide2000/ref/hdd/.

[4] Seagate disk drives. 2003. http://www.seagate.com/cda/products/discsales/index.

[5] M.J. Atallah and S. Prabhakar. (almost) optimal parallel block access for range queries. *Nineteenth ACM Symposium on Principles of Database Systems, PODS*, May 2000.

[6] R. Bhatia, Sinha R.K., and C.M. Chen. Declustring using golden ratio sequences. *In Proc. of International Conference on Data Engineering*, pages 271–280, February 2000.

[7] L. Richard Carley, Gregory R. Ganger, and David F. Nagle. MEMS-based integrated-circuit mass-storage systems. *Communication of the ACM*, 43(11), November 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[8] H.C. Du and J.S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Transactions of Database Systems*, 7(1):82–101, March 1982.

[9] C. Faloutsos and P. Bhagwat. Declustring using fractals. *In Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems*, pages 18–25, January 1993.

[10] K. Frikken, M. J. Atallah, Sunil Prabhakar, and R. Safavi-Naini. Optimal parallel i/o for range queries through replication. *In Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 669–678, September 2002.

[11] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle. Modeling and performance of MEMS-Based storage devices. *Proceedings of ACM SIGMETRICS*, pages 56–65, June 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[12] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle. Operating systems management of MEMS-based storage devices. *Symposium on Operating Systems Design and Implementation(OSDI)*, October 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[13] H.Yu, D.Agrawal, and A.El Abbadi. Tabular placement of relational data on MEMS-based storage devices. *In proceedings of the 29th Conference on Very Large Databases(VLDB)*, September 2003. To Appear.

[14] K.A.S.Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimensional data. *In International Conference on Database Theory*, pages 408–418, January 1997.

[15] S. Prabhakar, K.A.S.Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. *In International Conference on Data Engineering*, pages 94–101, February 1998.

[16] P.Vettider, M.Despont, U.Durig, W.Haberle, M.I. Lutwyche, H.E.Rothuizen, R.Stuz, R.Widmer, and G.K.Binnig. The ”millipede”-more than one thousand tips for future afm storage. *IBM Journal of Research and Development*, pages 44(3):323–340, May 2000.

[17] M. Riedewald, D. Agrawal, A. El Abbadi, and F. Korn. Accessing scientific data: Simpler is better. *In Proc. of the 8th International Symposium on Spatial and Temporal Databases*, pages 214–232, July 2003.

[18] Steven W. Schlosser, John Linwood Griffin, David F. Nagle, and Gregory R. Ganger. Designing computer systems with MEMS-based storage. *ASPLOS*, November 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[19] S.W. Schlosser, J. Schindler, A. Ailamaki, and G.R. Ganger. Exposing and exploiting internal parallelism in MEMS-based storage. *Technical Report CMU-CS-03-125, School of Computer Science, Carnegie Mellon University*, March 2003.

[20] B. Seeger. An analysis of schedules for performing multi-page requests. *Information Systems*, 21(5):387–407, 1996.

[21] B. Seeger, P.-A. Larson, and R. McFayden. Reading a set of disk pages. *In Proc. Int. Conf. on Very Large Databases (VLDB)*, pages 592–603, 1993.

[22] A.S. Tosun and H. Ferhatosmanoglu. Optimal parallel i/o using replication. *Proceedings of International Conference on Parallel Processing Workshops (ICPPW ’02)*, pages 506–513, August 2002.