New Hardware Support for Database Operations *

Nagender Bandi†, Chengyu Sun‡, Hailing Yu◇, Divyakant Agrawal†, Amr El Abbadi†
† Department of Computer Science, University of California, Santa Barbara
‡ Department of Computer Science, California State Universtity, Los Angeles
◇ Oracle Corporation, Redwood Shores, California

Abstract

Query processing cost in database applications can be divided into two parts: the I/O cost *and the* computational cost*. Traditionally DBMS researchers have focused on the issue of reducing the* I/O cost *as it is the primary source of bottleneck in most of the database operations. As the computer systems which run the DBMSs evolved, this problem worsened because the speeds of secondary storage devices increased at a much slower pace compared to the memory access and processing speeds. Further, as databases become increasingly accepted in areas such as spatial databases, we encounter new challenges. For example, the* computational cost *has been known to be the bottleneck in certain spatial operations. In this paper, we present a survey of our work where we present novel solutions which use hardware to alleviate the* I/O cost *bottleneck for traditional database applications and the* computational cost *bottleneck for non-standard database applications such as spatial databases. In particular, we use the novel MEMS-based storage devices to develop efficient data placement solutions for relational databases and two-dimensional datasets. Using commodity graphics hardware, we develop novel spatial solutions which outperform conventional software techniques. We also integrate these techniques into a popular commercial spatial database and demonstrate significant improvement in query performance.*

1 Introduction

The cost of a DBMS query consists of two factors: *I/O cost*, the time spent in loading the data from the secondary storage into the main memory, and *computational cost*, the time spent by the DBMS in processing this data and returning the result. Traditionally *I/O cost* has been known to be the major bottleneck due to the high cost of disk access and algorithms were developed to reduce the I/O cost [10]. Although the past two decades witnessed tremendous improvement in the processing and memory speeds, the disk access speeds have not substantially changed thus worsening the I/O cost bottleneck. Furthermore, increasing acceptance of DBMSs in non-traditional areas, such as spatial databases, presents new interesting challenges. For example, *computation cost* accounts for most of the query cost for certain spatial operations [12]. Although many solutions have been proposed over the last decade to alleviate this *computation cost*, the problem still persists. Most of these solutions have been software based, however, recently, more interest has been directed to using the impressive

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research has been supported by the NSF grant IIS-02-20152.

advances in hardware to reduce both the I/O and the computational costs. We believe that novel architectures and hardware devices promise to provide efficient solutions to some of the significant bottlenecks encountered by DBMSs. In this paper, we present a survey of our work which provide efficient hardware-based solutions that alleviate the *I/O cost* in the case of relational databases and *computation cost* in spatial applications.

In Section 2, we present efficient data placement and retrieval solutions for relational databases and two-dimensional datasets. Our research is based on the novel MEMS-based storage devices [1, 18, 2] being developed towards replacing magnetic disks as secondary storage media. We exploit the inherent two-dimensional layout of the MEMS-based storage to develop a data placement scheme for relational databases which caters for both OLAP and OLTP applications in most efficient way [19]. We also apply the high bandwidth parallel transfer provided by thousands of synchronous tips of a MEMS-device to develop mappings for two-dimensional datasets which enable highly optimal retrieval for window queries [20].

In Section 3, we discuss novel spatial database techniques [17, 6] where we exploit the computation power of off-the-shelf graphics hardware towards achieving significant performance improvement over most sophisticated techniques. We also integrate our techniques into a popular commercial database and show that significant reduction in the computation cost can be achieved through this integration.

2 MEMS-based Storage

In this section, we describe the basic features of MEMS and how we use MEMS to develop efficient data placement algorithms for relational databases and two-dimensional datasets. MEMS are extremely small mechanical systems formed by the integration of mechanical elements, actuators, electronics, and sensors. They are fabricated on silicon chips using photo-lithographic processes similar to those employed in manufacturing standard semiconductor devices. As a result, MEMS-based storage can be manufactured at a very low cost. They represent a compromise between slow traditional disks and expensive storage based on EEPROM technologies. Due to the great potential for applications MEMS present, several major research labs such as IBM [18] and HP [2], and universities such as Carnegie Mellon University [1] have been involved in their development. MEMS devices from these research groups differ largely in the types of actuators used to position the media and the media used to record data in the medium. Figure 1 shows a high level view of a MEMS-based storage device from CMU CHIPS project [15]. The media sled is organized into rectangular regions at the lower level. Each of these rectangular regions contains over millions of bits and is accessible by one tip. Because of heat-dissipation constraints, in the CMU CHIPS model, not all tips can be activated simultaneously to access data even though it is theoretically feasible.

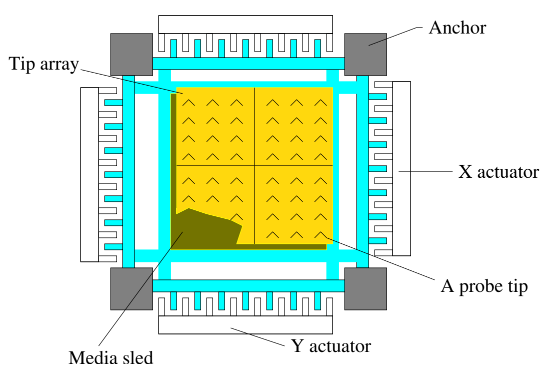

Figure 1: The architecture of CMU CHIPS

MEMS-based storage has several advantages over traditional magnetic disk technology. Since MEMS do not contain large mechanical components, the access latency is significantly smaller. Furthermore, since MEMS are based on conventional silicon-based technology, they can leverage from the large-scale integration resulting in much smaller form factor and economies of scale. Another advantage of MEMS over disks is that the power consumption is much smaller and they do not incur long latencies when powered-up or powered-down. Recent efforts in computer science research have focused on taking advantage of MEMS-based storage devices from different perspectives. One such aspect is how to integrate MEMS-based storage into current systems and to develop novel solutions using these devices. In our studies, we exploit the inherent characteristics of MEMS-based storage devices and design novel data layout designs for different applications [19, 20]. In particular, we exploit the property of MEMS-based storage devices that they are inherently two-dimensional and can provide high-bandwidth concurrent data access through thousands of concurrent tips. In the following subsections, we describe our data placement schemes for relational databases and two-dimensional datasets.

2.1 MEMS-based Storage Architecture for Relational Databases

Databases have always suffered from the increasing gap between economically viable non-volatile hard disks and expensive but fast transistor-based in-memory chips. One of the main challenges in database research is to find efficient methods to support operations that have conflicting needs in terms of storage and page layouts. In particular, due to the update characteristics of OLTP transactions, relations need to be accessed in a row-wise manner. In contrast, for OLAP queries where only a subset of attributes are of interest, the I/O system should facilitate data retrieval in a column-wise manner. Hence OLTP workloads perform better with row-wise placement of relations while OLAP workloads perform better with column-wise allocation. However the traditional data layouts over disk devices for relational data neither favor the OLTP applications nor the OLAP applications. Since a disk is modeled as an one-dimensional array, achieving row-wise access and column-wise access are conflicting goals. Recent proposals have been made to address this problem at both the cache and disk levels. Ramamurthy et al. [14] propose multiple representations of the same relation on disks to support such conflicting requirements. Even though multiple representations of the same relation alleviates this problem, two copies of each relation have to be stored. In order to address cache performance, Ailamaki et al. [4] propose PAX that reorganizes the data in each page so as to reduce cache pollution. The essence of PAX is not to pollute cache with unnecessary data, but no I/O time is saved.

Based on these observations, we take advantage of the inherent properties of MEMS-based storage and propose to replace disks in relational database systems. In [19], we propose a new strategy for placing relational data on MEMS-based storage devices [19]. The new data placement scheme exploits the two-dimensional nature of MEMS-based storage devices to support efficient query processing for different types of workloads. In other words, the new scheme supports both row-wise and column-wise access to relations, hence the unnecessary data will not be brought into main memory and a lot of I/O time is saved. We also exploit the parallel retrieval property of MEMS devices to efficiently reduce cache pollution. As a result, our data placement strategy for placing relations on MEMS-based storage improves I/O time as well as reduces the processor and main memory gap. Our theoretical analysis and experimental results show that the new strategy can result in significant improvements for relational DBMSs. The TPC-H queries can be improved up to 5 times compared to disk devices. We also develop query processing algorithms for different types of queries and address the issues of indexes, because MEMS-based storage devices have a different interface from disks in our design.

2.2 Slicing and Dicing Two-dimensional Datasets

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 datasets has been rapidly growing, hence, fast retrieval of relevant data is the key to improve query performance. 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 [3], it is shown that strictly optimal solutions for range queries exist in only a small number of cases. However, all prior research, including the optimal cost formulation itself, is 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 [20], we define the new optimal cost and prove that it is impossible to achieve the new optimal cost for the general case, 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 order to improve the performance of range queries over two-dimensional datasets, we explore data placement schemes for two-dimensional data over MEMS-based storage to facilitate efficient processing of this type of queries. 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. We utilize the Y-axis sequential access of MEMS-based storage devices to achieve the Y dimensional sequential access over a dataset and take advantage of intra-device parallelism to achieve the X dimensional sequential access. The new scheme allows for the reorganization of query ranges into new tiles which contain more relevant data, thus maximizing the number of concurrent active tips to access the relevant data. In our new scheme, we tile two-dimensional datasets according to the physical properties of I/O devices. In all previous works, 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 do not need to know how or where data is stored. Our theoretical analysis and experimental results show that our new scheme can achieve almost optimal performance.

3 Hardware Acceleration for Spatial Database Operations

In this section, we address the problem of alleviating the computational cost bottleneck for spatial database applications. We observe that spatial database operations lend themselves well to hardware acceleration. Specifically, both spatial databases and graphics hardware abstract real world objects as geometries such as points, polylines, and polygons, and process them based on their topological properties and relationships. In our recent work [17, 6], we developed a hardware intermediate filter for spatial database primitives, and showed that this technique can be easily integrated into commercial databases without changes to existing storage and index structures.

3.1 Spatial Query Evaluation

Spatial database queries are typically evaluated in two steps: the *filtering step* and the *refinement step*. In the filtering step, the minimal bounding rectangles (MBRs) of the objects and spatial indexes such as R-trees [11] are used to quickly determine a set of candidate results. In the refinement step, the final results are determined by retrieving the actual geometries of the candidates from the database, and comparing them to either a query geometry or to each other. For complex geometries such as polygons, the cost of the refinement step usually dominates the query cost due to the complexity of the underlying computational geometry algorithms.

The cost of the refinement step consists of two factors: the *I/O cost* of loading the geometries from disk to main memory, and the *computational cost* of geometry-geometry comparison. The ratio of the computational cost over the I/O cost varies significantly depending on the types of spatial queries and the complexity of the geometries, which can be roughly characterized by the number of vertexes of a geometry. Generally speaking, the more complex the data, the more significant the computational cost. For instance, a recent study on spatial selections [12] shows that for point geometries, the I/O cost is the dominant factor, but for polygon geometries, both costs are significant. In the case of a spatial join, the computational cost could be orders of magnitude higher than the I/O cost, because once a geometry is loaded, it is buffered in the main memory and compared to many other geometries.

(a) LANDC (b) LANDO

Figure 2: Sample Objects from Two Dataset

The high computational cost of the spatial operations comes from the complexity of the data in the real world, where it is not uncommon that a polygon has tens of thousands of vertexes. Furthermore, the shapes of the polygons can be arbitrarily complex, as can be seen from Figure 2, which shows the first 100 polygons in the Wyoming land cover dataset (LANDC) and the Wyoming land ownership dataset (LANDO). In many cases, the polygons are concave, and sometimes, even non-simple[1]. Processing these types of polygons is very expensive. For instance, assuming the numbers of vertexes of two polygons are n and m, the complexity of the commonly used intersection test algorithm is $(O((n+m)\log(n+m))$ [9], and the distance calculation algorithm is even more expensive with $O(n \times m)$ worst case complexity [8]. Since the I/O cost remains linear, the computational cost quickly outweighs the I/O cost as the complexity of data increases.

Over the last decade, many techniques have been proposed to improve spatial query processing, and most of them rely on an *intermediate filtering* step [7, 21, 5, 12, 8], where false or positive hits can be identified without performing the costly geometry-geometry comparison. These techniques have been shown to be very effective for improving query performance, but suffer from two drawbacks. First, the computation required to pre-process the data is usually expensive, which degrades update performance, and cannot be applied to cases where one of the datasets is an intermediate dataset, e.g. the results of a map overlay operation. Second, the pre-computed polygon approximations or auxiliary data structures have to be stored on disk. The extra storage and I/O costs may not be significant, but it still requires changes of existing storage or index structures, therefore it is more difficult to incorporate these techniques into commercial DBMSs [13].

3.2 Hardware-based Intermediate Filtering

Our technique is based on the observation that most low-level algorithms spatial databases rely on have been well studied by the computational geometry community. Under current computer architectures, it is unlikely that algorithmic advances will significantly reduce the cost of geometry-geometry comparison. On the other hand, the last few years have seen tremendous advances in graphics hardware technologies. Off-the-shelf graphics cards are now capable of handling thousands of polygons in real time, and are widely used in computer games, 3D modeling, and virtual reality applications. Since both graphics hardware and spatial databases work on geometries such as points, lines, and polygons, and they both deal with geometric relations such as intersection and containment in a 2D or 3D space, it is only natural to exploit the computational power of graphics hardware to speed up spatial database operations.

The general strategy to test for intersection of two polygons with graphics hardware is quite straight-forward:

- Render the first polygon with color c_1
- Render the second polygon with color c_2

[1]Non-simple polygons are polygons with self-intersecting edges or with vertexes that have degrees greater than 2 (more than 2 edges incident to a vertex).

- Search the frame buffer for overlapping pixels with color $c_1 + c_2$. If such pixels exist, these two polygons intersect each other.

An example of this strategy is illustrated in Figure 3(a), where the two polygons are rendered with color *gray*, and the overlapping pixels are *black*.

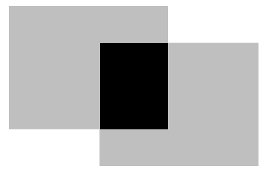

Figure 3: Hardware Intersection Test

It turns out that implementing this strategy is quite difficult. First of all, in order to achieve maximum performance for the most common tasks, graphics hardware is designed to support only a limited set of geometries, which include points, line segments, and convex polygons. Concave polygons, which are common in spatial datasets, must be triangulated before they can be rendered by hardware. Polygon triangulation, which has to be performed in software, is much more expensive than hardware operations. Secondly, the coordinates of publicly available GIS data typically have 4 to 6 digits accuracy, or about 11 to 18 bit accuracy in binary form. On the other hand, the resolution of a rendering window is rather limited. For instance, a 32×32 rendering window only provides 5 bit accuracy in each dimension. This disparity between the data resolution and the window resolution means that the hardware approach may produce *false* results, which are usually not acceptable for spatial database applications.

Taking these issues into account, we developed a polygon intersection test algorithm [17] that uses hardware segment intersection test as an additional filtering step. This algorithm has several properties:

- The algorithm only renders the boundary of a polygon, thereby is highly efficient regardless of the convexity of the polygon.

- The algorithm guarantees no false results based on the anti-aliasing property of OpenGL [16],

- The algorithm can be easily extended to handle distance test.

3.3 Integration with Oracle Spatial

In order to evaluate the performance of the hardware-based spatial solutions against sophisticated software-based techniques, we integrated our techniques into Oracle, a popular commercial spatial database [6]. We developed a general framework for integrating novel hardware-based techniques such as ours into a commercial database. We integrate the hardware-accelerated spatial solutions as external procedures where the query processing which accesses the graphics hardware happens in a process external to the address-space of the DBMS. This integration enables a comparison of the run-time hardware-based techniques against both run-time and preprocessing-based software techniques provided by Oracle. It also presents how our technique seamlessly integrates with existing software-based indexes with out requiring any modifications as in the case of preprocessing-based solutions.

3.4 Performance Results

Experimental evaluation over real world datasets show that the hardware-accelerated spatial operations achieve significant performance improvements over sophisticated software-based algorithms. Comparison of hardware-based intersection solutions against run-time software solutions [6] using R-tree indexes show an average case improvement of 3-4 times for spatial selection queries. In the case of spatial joins, we achieve an improvement of 3-4 times for intersection joins and 5.9 times for with-in distance joins. We also compare the performance of the hardware-based solutions with preprocessing-based software solutions. Preprocessing-based solutions in Oracle utilize Quad-tree indexes and the preprocessing accuracy is determined by the Quad-tree tiling level. Preprocessing cost increases exponentially with the tiling level both in terms of index creation time and index storage cost where a typical index creation for a real world megabyte-sized dataset can take hours of creation cost and gigabytes of index space. Our evaluation shows that the hardware-based solutions not only outperform the preprocessing-based solutions at lower tiling levels but also performs as well if not better at higher tiling levels with out incurring any of the preprocessing overheads.

4 Conclusion

In this paper, we briefly highlighted some of our recent research exploiting modern hardware to improve the overall performance of DBMS operations. Software solutions are restricted in terms of how much improvements they can provide. Hardware, on the other hand, has been achieving incredible advances in recent years. In our MEMS work, we are exploiting a new and futuristic technology, with the goal of reducing the I/O cost of retrieving data. As material and mechanical engineers develop new and more efficient MEMS-based storage devices, we believe the role of the computer scientist, in general, and the database researcher, in particular, is to develop efficient ways to integrate such devices into computer systems. Our work represents an initial step in this direction for data placement, and has shown the great potential MEMS has for computer systems. On the other hand, our research with graphics hardware exploits a well established and widely available hardware device. In fact, due to its widespread availability on most modern day computers, we strongly believe that graphics cards should be exploited as much as possible to improve the performance of large software systems, and in particular databases. Our research has clearly demonstrated the successful integration of graphics hardware with commercial databases, resulting in substantial performance improvements. We are continuing, at UC Santa Barbara, to explore such trends, namely using both futuristic devices with high potential, as well as existing off the shelf hardware for new and novel applications.

References

[1] CMU CHIP project. 2002. 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] 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.

[4] A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis. Weaving relations for cache performance. *In proceedings of the 27th Conference on Very Large Databases*, pages 169–180, September 2001.

[5] Wael M. Badawy and Walid G. Aref. On local heuristics to speed up polygon-polygon intersection tests. In *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems*, pages 97–102, 1999.

[6] Nagender Bandi, Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Hardware acceleration in commercial databases: A case study of spatial operations. In *Proceedings of 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 1021–1032, 2004.

[7] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 197–208, 1994.

[8] Edward P.F. Chan. Evaluation of buffer queries in spatial databases. In *Proceedings of 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*, pages 161–170, July 2001.

[9] Mark De Berg, Marc Van Kreveld, Mark Overmars, and O. Scharzkopf. *Computational Geometry: Algorithms and Applications*, chapter 2. Springer, 2000.

[10] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database System Implementation*, chapter 4. Prentice Hall, 1999.

[11] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57, 1984.

[12] Ravi K. Kothuri and Siva Ravada. Efficient processing of large spatial queries using interior approximation. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*, pages 404–421, 2001.

[13] Ravi K. Kothuri, Siva Ravada, and Daniel Abugov. Quadtree and r-tree indexes in oracle spatial: A comparison using gis data. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of DATA (SIGMOD'02)*, pages 546–557, 2002.

[14] R. Ramamurthy, D.J. DeWitt, and Q. Su. A case for fractured mirrors. *Proc. Int. Conf. on Very Large Data Bases*, pages 430–441, September 2002.

[15] S. W. Schlosser, J. L. Griffin, D. F. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 1–12, 2000.

[16] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.2.1)*. Silicon Graphics, Inc., April 1999.

[17] Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Hardware acceleration for spatial selections and joins. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 455–466. ACM Press, 2003.

[18] 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*, 44(3):323–340, May 2000.

[19] H. Yu, D.Agrawal, and A. El Abbadi. Tabular placement of relational data on MEMS based storage devices. In *Proc. Int. Conf. on Very Large Data Bases*, pages 680–693, 2003.

[20] H. Yu, D.Agrawal, and A. El Abbadi. Declustering two-dimensional datasets over MEMS based storage. In *9th International Conference on Extending DataBase Technology*, pages 495–512, 2004.

[21] Geraldo Zimbrao and Jano Moreira de Souza. A raster approximation for processing of spatial joins. In *Proceedings of 24rd International Conference on Very Large Data Bases (VLDB'98)*, pages 558–569, 1998.