Report Number: WUCSE-2002-23

2002-08-01

Exploiting Truly Fast Hardware in Data Mining

Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck

In many data mining applications, the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off magnetic media, cross the system bus into main memory, copy into processor cache, and then execute code to perform a search is prohibitive. We are proposing that a significant portion of the data mining task (i.e., the portion that examines the bulk of the raw data) be implemented in fast hardware, close to the magnetic media on which it is stored. Furthermore, this hardware can be... Read complete abstract on page 2.

Exploiting Truly Fast Hardware in Data Mining

Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck

Complete Abstract:

In many data mining applications, the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off magnetic media, cross the system bus into main memory, copy into processor cache, and then execute code to perform a search is prohibitive. We are proposing that a significant portion of the data mining task (i.e., the portion that examines the bulk of the raw data) be implemented in fast hardware, close to the magnetic media on which it is stored. Furthermore, this hardware can be replicated allowing mining tasks to be performed in parallel, thus providing further speed up for the overall mining application. In this paper, we describe a general framework under which this can be accomplished, and identify a number of important research questions that must be addressed for it to be practical.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2002-23

Exploiting Truly Fast Hardware in Data Mining

Authors: Chamberlain, Roger D.; Cytron, Ron K.; Franklin, Mark A.; Indeck, Ronald I.

August 1, 2002

Abstract: In many data mining applications, the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off magnetic media, cross the system bus into main memory, copy into processor cache, and then execute code to perform a search is prohibitive. We are proposing that a significant portion of the data mining task (i.e., the portion that examines the bulk of the raw data) be implemented in fast hardware, close to the magnetic media on which it is stored. Furthermore, this hardware can be replicated allowing mining tasks to be performed in parallel, thus providing further speedup for the overall mining application. In this paper, we describe a general framework under which this can be accomplished, and identify a number of important research questions that must be addressed for it to be practical.

Exploiting Truly Fast Hardware in Data Mining

Roger D. Chamberlain
Ron K. Cytron
Mark A. Franklin
Ronald S. Indeck

Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck, "Exploiting Truly Fast Hardware in Data Mining," Technical Report WUCS-2002-23, Dept. of Computer Science and Engineering, Washington University, St. Louis, MO, August 2002.

Department of Computer Science and Engineering
Washington University
Campus Box 1045
One Brookings Dr.
St. Louis, MO 63130-4899

Exploiting Truly Fast Hardware in Data Mining

R. D. Chamberlain[*], R. K. Cytron[*], M. A. Franklin[*], and R. S. Indeck[†]
[*]Department of Computer Science and Engineering
[†]Department of Electrical Engineering
Washington University, St. Louis, Missouri

Abstract

In many data mining applications, the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off magnetic media, cross the system bus into main memory, copy into processor cache, and then execute code to perform a search is prohibitive. We are proposing that a significant portion of the data mining task (i.e., the portion that examines the bulk of the raw data) be implemented in fast hardware, close to the magnetic media on which it is stored. Furthermore, this hardware can be replicated allowing mining tasks to be performed in parallel, thus providing further speedup for the overall mining application. In this paper, we describe a general framework under which this can be accomplished, and identify a number of important research questions that must be addressed for it to be practical.

1. Introduction

Having data storage so affordable has enabled the amassing of vast amounts of information. These data are rapidly accessible, motivating a significant interest in data mining capability. At present, these data sets far exceed the capacity of modern processors, so searching them has become a serious challenge. In a recent invited talk [1] at the High Performance Embedded Computing Workshop, John Reynders of Celera Genomics commented that, "The size of the databases we deal with is no longer measured in terabytes, but in exabytes."

In addition to being quite large, database sizes are also growing at an unbelievable pace. The average database size and associated software support systems are growing at rates that are greater than the increase in processor performance (i.e., doubling more than every 18 months). This is due to a number of factors, including the desire to store more detailed information, to store information over longer periods, to merge databases from disparate organizations, and to deal with the large new databases that have arisen from emerging important applications. For example, two emerging applications (in the civilian domain) having large and rapidly growing databases are those connected with the genetics revolution and those associated with cataloging and accessing information on the Internet. At the physical level this has been made possible by the remarkable growth in disk storage performance where magnetic storage density has been doubling every year for the past several years.

Estimates are that in excess of 1.5 million web pages are added to the Internet each day. Companies that operate search engines have a difficult time keeping up with the torrent information that requires indexing. Today, the performance bottleneck is the time taken to develop the required reverse index in reasonable time. Quoting from Domingos and Hulten [2]:

> In a single day WalMart records 20 million sales transactions, Google handles 70 million searches, and AT&T produces 275 million call records. Current algorithms for mining complex models from data (e.g., decision trees, sets of rules) cannot mine even a fraction of this data in useful time. Further, mining a day's worth of data can take more than a day of CPU time, and so data accumulates faster than it can be mined.

Further complicating the situation is the fact that many database mining operations require searching a large unstructured space for approximate matches, and unstructured data that is rapidly changing is ill suited to reverse indexing. The two examples given above, genetics and the Internet, are illustrative of this. These same factors are at the root of the problems associated with extracting useful intelligence from image data as well. The problem here is indeed even more difficult since the data tends to be very unstructured and the keys used in indexing are continuously evolving over time.

Currently, data mining applications are implemented using traditional, off-the-shelf hardware platforms. Figure 1 illustrates the relevant features of virtually all of these systems. A disk (actually many disks) is attached via a controller to the I/O bus of a computer system. A path (labeled "bridge" in the figure) exists that enables data to flow from the I/O bus to the memory bus (and therefore to the system's main memory). When an algorithm is executed on the processor, references to the main memory cause the data to be loaded into cache, at which point the processor can efficiently access the data.

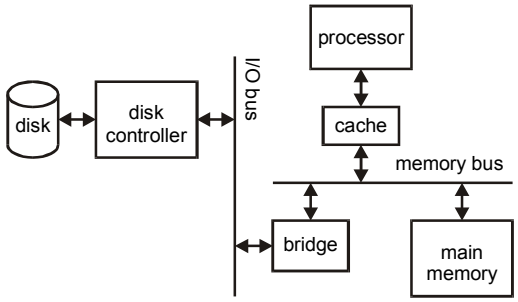

Figure 1. Typical hardware platform for data mining applications.

Add the operating system overhead to the above data movement requirements, and it is clear that there are significant data movement inefficiencies in current systems. Yet this is the system environment associated with development and deployment of virtually all of today's data mining applications. The result is that even though individual components in the system are quite fast (e.g., modern processors have clock speeds approaching 2 GHz), the overall performance suffers because these fast components are used inefficiently.

We propose to dramatically increase the speed with which large volumes of data can be mined by eliminating the above inefficiencies and mining data much closer to where it resides, on the disk, and by performing low-level mining operations directly in reconfigurable hardware. Additionally, by replicating this hardware on a per disk surface or per disk subsystem level, we can exploit the parallelism inherent in most search based algorithms to provide speedups that scale linearly with the reconfigurable hardware deployed. The result is a system that is truly fast, since the individual components are efficiently used. Preliminary estimates are that, for important applications, almost two orders of magnitude speedup can be achieved.

In this paper, we describe a general framework under which this can be accomplished, and identify a number of important research questions that must be addressed for it to be practical. Section 2 describes the system architecture, including how it performs basic mining operations without the overhead of data movement across the I/O bus and memory bus and into the processor. Section 3 discusses two distinct application domains that could see significant benefit from this type of system. Section 4 describes the research issues necessary for the system to perform effectively on a general set of database mining problems, and Section 5 provides a summary and conclusions.

2. System Architecture

The proposed system architecture is illustrated in Figure 2. Associated directly with a disk head is a Data Shift Register (DSR) that receives data streaming off the head at disk rotational speeds. The data in the DSR is made available (in parallel form) to reconfigurable logic that performs low-level mining operations on the data that has been retrieved off the disk. The specific function performed by the reconfigurable logic will be tailored to the particular data mining application of interest. Example functions for a pair of applications are described below in Section 3.

Also present in the system is a microprocessor that is used both for control duties (e.g., replicating the function of the disk controller in Figure 1 and managing the function of the reconfigurable logic) and higher-level data mining operations. The remainder of the system reflects traditional designs, with an I/O bus, a bridge to the memory bus, and a classic memory hierarchy. The host processor is still responsible for managing the file system and maintaining the general functionality of the database, the new hardware is used primarily for high-volume mining operations.

In the illustration of Figure 2, only one copy of the DSR and reconfigurable logic is shown; however, the intent is to replicate both for each head in the disk, providing significant additional parallelism over what is typically available when accessing the disk head via standard interfaces (e.g., ATA or SCSI). The result is a system that can perform relatively complex data mining operations across a high volume of data directly at hardware speeds, eliminating the overheads associated with the I/O bus, the memory bus, and the operating system.

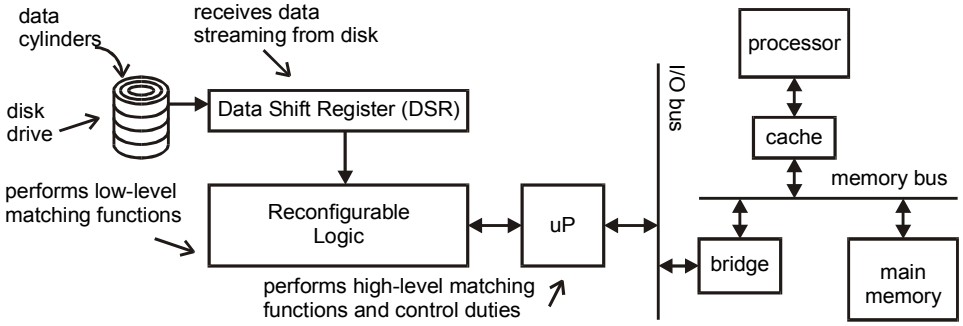

Figure 2. System architecture.

In the description that follows, we assume (for discussion purposes) that the data is unstructured text and we are performing string matching queries. Realistic queries are generally compound in nature. In this case the search involves both matching the query strings to documents on the disk, and determining if the relationships exist when matches occur (or don't occur). For example, a query might contain the strings "Iraq", "Iran", "Israel" and "France", and want to identify documents where either "Iraq" ***and*** "Israel", ***or*** "Iran" ***and*** "France" are present. This form of compound inquiry may result in a number of document matches, and the objective is to perform these matches at disk speeds. In general, such queries can be represented as a tree structure where the leaves of the tree correspond to the byte strings being sought (e.g., Iraq, Israel, etc.), and the nodes correspond to the logical operations required of the query. One way of viewing the processing of a compound query is in terms of two components:

- processing of the leaves associated with the bottom of the tree (i.e., taking the leaf words and performing a match with data on the disk), and
- performing the logic operations associated with the combining nodes in the query tree structure.

In the proposed architecture, leaf processing (word matching) is done in the reconfigurable hardware (at disk speeds) and the results are sent to the dedicated control microprocessor that acts to execute the logic associated with the nodes of the query tree. The result of this logic execution is a set of results that are sent to the host processor that initiated system activity by sending the original query.

3. Example Applications

To illustrate the use of the system, two example applications are described: an unstructured text search and biological sequence matching. Both applications are currently heavily used, and both tax the capabilities of current systems to deliver the throughput desired.

Unstructured text searching: As described above, a compound query posed in a text search context can be decomposed into the individual word searches (to be executed in the reconfigurable logic) and the composition of the word search results (executed on the microprocessor). Here, we describe a simple mechanism whereby the individual word searches can return positive results for approximate matches.

An example configuration for the reconfigurable hardware is illustrated in Figure 3. At the top of the figure is a section of the DSR that contains raw data streaming off the disk head. Immediately below the DSR is a Compare Register (CR) that contains pattern data. Next is Fine-Grained Comparison Logic (FCL) that performs element by element comparisons between elements of the DSR and the CR. The FCL can be configured to be either case sensitive or case insensitive. Also notice the alternate routing of DSR elements to individual FCL cells on the right-hand side of the figure. An FCL cell that can match more than one position in the DSR enables a single 6 element CR to match both the commonly used spelling of "Baghdad" as well as the alternate "Bagdad" in shared hardware. Below the FCL cells is Word-Level Comparison Logic (WCL) that is responsible for determining whether or not a word match occurs. The actual configuration will vary with query type. The Word-Level Match Signals are

delivered to the control microprocessor for evaluation of the compound query. A match to the entire query is reported to the host processor. A VHDL design of this example, configured for exact matching, synthesizes to a FPGA-based gate-level equivalent model that is estimated to execute at over 250 MHz, for an aggregate data throughput of 2 Gb/s [3]. This is in contrast to measured throughput data for the Unix `grep` utility (executing the software-level equivalent function) of approximately 200 Mb/s.

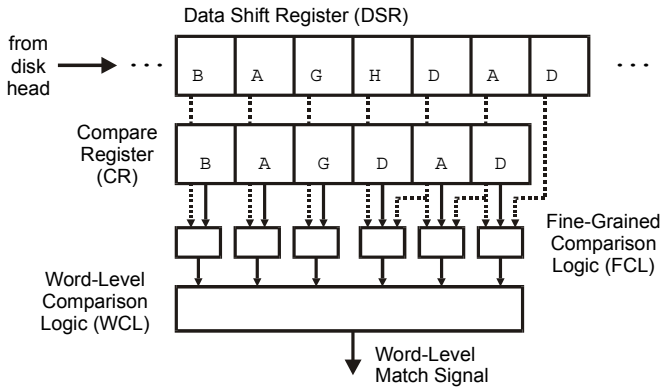

Figure 3. Reconfigurable hardware for text search application.

Sequence matching: The basic set of operations in sequence matching corresponds to a dynamic programming problem when executed on a conventional system [4,5]. This can be described using Figure 4. Here, $p_1p_2p_3p_4$ represent 4 symbols from a pattern *p,* $t_1t_2t_3 \ldots t_9$ represent symbols from a target *t*, and $d_{i,j}$ represents the edit distance at position *i* in the pattern and position *j* in the target. In normal usage the pattern is short relative to the target. Typical sizes might have *p* on the order of 1000-2000 characters and *t* many billions of characters.

The dynamic programming problem is as follows. There is a set of known (constant) values for an additional row (d0,*j*) and column (d*i*,0) not shown in the figure. Additionally, there are two "scoring functions" provided, A and $B_{i,j}$, where A is a constant and $B_{i,j}$ is a function of p_i and t_j (i.e., $B_{i,j} = f(p_i, t_j)$).

The values for $d_{i,j}$ are computed using the fact that $d_{i,j}$ is only a function of the following set

$$\{p_i \quad t_j \quad d_{i-1,j-1} \quad d_{i-1,j} \quad d_{i,j-1}\}.$$

This is illustrated in the figure by showing the dependency of $d_{3,6}$ on the values of $d_{2,5}$ and $d_{2,6}$ and $d_{3,5}$ as well as p_3 and t_6 (through A and B). The form of the function can be quite arbitrary, but is usually constructed in the following form:

$$d_{i,j} = \max[d_{i,j-1} + A;\ d_{i-1,j} + A;\ d_{i-1,j-1} + B_{i,j}].$$

A match is declared when a value in the table exceeds a predetermined threshold. The match is then examined via a traceback operation within the table. In the proposed architecture, the reconfigurable logic computes the entries in the table ($d_{i,j}$) and checks for the threshold. The control microprocessor is responsible for the traceback.

Figure 5 shows the block diagram of a systolic array architecture for computing the values in the table. The characters of the pattern are stored in the column of registers on the right of the figure (labeled p_1, p_2, p_3, and p_4). The characters in the target flow from left to right in the shift registers at the top of the figure (labeled t_5 through t_8, illustrating positions 5 through 8 in the target). The systolic cells are in the interior of the figure (labeled $d_{i,j}$). In the first (combinatorial) part of a clock cycle, the four underlined values are computed. For example, the new value $d_{3,6}$ is shown to depend upon the same five values illustrated earlier in Figure 4. In the second (latch) part of the clock cycle, all the characters in *t* and *d* are shifted one position to the right. In an initial VHDL design of the systolic array (targeting a Xilinx FPGA implementation) it is estimated that the reconfigurable logic will be capable of executing in excess of 80 MHz; however, this design is not yet optimized, and we expect that a fully optimized design to execute with a throughput of 4 times that of the initial design [3].

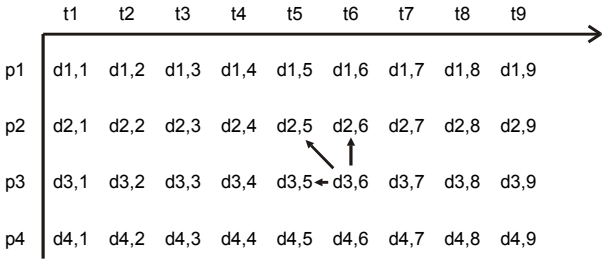

Figure 4. Dynamic programming example.

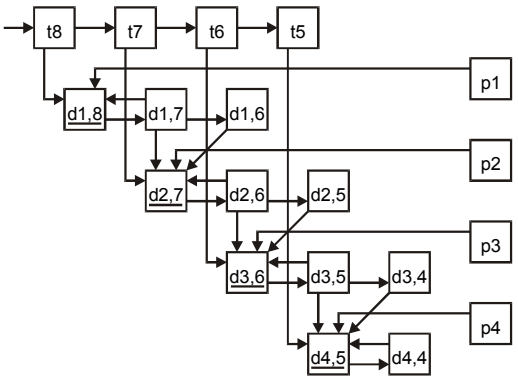

Figure 5. Systolic array for sequence matching.

Other applications: The above section has described the reconfigurable hardware structure and operation for two specific applications. Other data mining applications will have distinct low-level operations that need to be performed. One strength of this system is the fact that the reconfigurable logic is flexible enough to support not only the two example applications already described, but a large variety of additional structures, even those not yet envisioned when the system is designed. Just how to design and implement new applications is the focus of the research directions described in the following section.

4. Research Questions

A general pattern that can be extrapolated from the two example designs given above is that the reconfigurable hardware is well suited to fast, simple operations. It basically acts as a first-level data filter, examining a large volume of data quickly, and informing the higher-level system what subset of the data is worthy of further consideration. In essence, the hardware is very good at low-level matching operations. Database mining, however, is less about low-level matching operations and more about high-level functionality. More and more sophisticated data models are employed, and we ask more and more complex questions of the data. This high-level functionality is not particularly well suited to direct hardware implementation.

The above observation points to a hierarchical approach to the use of the system. The reconfigurable logic is used to perform low-level operations, pre-filtering the data at high throughput rates, and software is used to perform high-level operations, implementing sophisticated data models and answering complex queries. In the most general case, an arbitrary query will go through the transformation process illustrated in Figure 6. We assume that the database system that executes on the host processor generates a query (representing some data mining operation). This query proceeds through a compiler (second block), also located on the host processor, which is responsible for query analysis. There are two main results from the query analysis:

- determining the structure of the reconfigurable logic required to perform low-level operations on the data streaming off the disk, and
- determining the high-level operations that must be implemented in the control microprocessor.

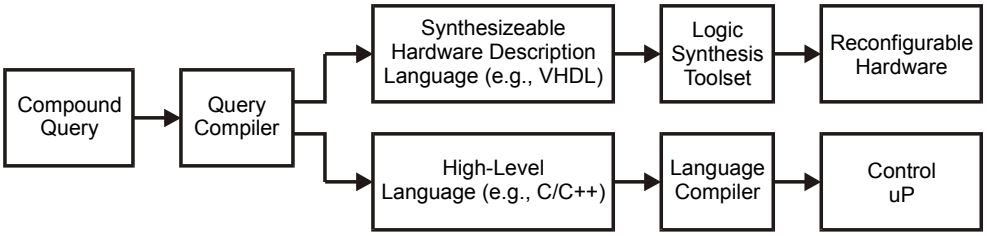

Figure 6. Transformation of queries for data mining.

Although the path represented by Figure 6 is quite general, and will therefore be able to handle a wide range of potential queries, it has the drawback that the latency introduced into the search process might be too long. If the compilation process is too long, it might become the performance bottleneck rather than the search itself. This issue

can be addressed for a wide range of likely queries by maintaining a set of precompiled hardware templates that handle the most common cases.

The research questions that warrant investigation relate to how one can effectively use the system described above to significantly accelerate the speed at which data mining can take place. We divide the set of research questions into two groups, file system questions and query compiler questions.

File System Questions: If the above described hardware system is to be effectively utilized, the file system must be organized in such a way that data can continually stream off the disk head and into the data shift register. This requires a different set of data placement techniques than are currently used. In addition, the data placement algorithms must have access to true physical CHS (cylinder, head, sector) information. It is current practice to address a disk drive with logical CHS information, which is then mapped internally by the drive controller to an arbitrary (typically unreported) physical CHS. To support true streaming data, minimizing seek time and inter-block gaps, we must have access to the logical to physical CHS mapping.

Query Compiler Questions: There are a host of important issues that must be addressed in this area. Examples include:

- At what semantic level, and using what languages and/or APIs, can queries be specified? This clearly must be tied into the types of operations that database mining requires now and will likely require in the future.
- How are the queries decomposed into low-level operations (for execution directly in hardware) and high-level operations (for execution in software)?
- What is an appropriate set of presynthesized hardware templates that would diminish the frequency with which the full compilation process represented by Figure 6 must take place?

As we gain experience with the use of reconfigurable hardware, many additional research questions will invariably surface.

5. Summary and Conclusions

This paper presents the basic design of a data-mining system that has the potential for truly fast operation, unhindered by the overheads imposed by the I/O bus, main memory bus, cache, operating system, etc. An important requirement for the ultimate success of this system is the decomposition of data mining operations into low-level operations that can execute on the reconfigurable hardware and high-level operations that execute in software. We have posed several research questions, the answers to which will help meet this requirement.

We currently have initial prototype implementations of the reconfigurable hardware for the two applications described above, and are working to further improve their performance. Those designs are described in more detail in [3]. We are also developing performance models that help assess the performance gains that can be achieved using the proposed system.

[1] John Reynders, "Computing Biology," invited talk at 5th High Performance Embedded Computing Workshop, November 2001.

[2] Pedro Domingos and Geoff Hulten, "Catching Up with the Data: Research Issues in Mining Data Streams," in Workshop on Research Issues in Data Mining and Knowledge Discovery, May 2001.

[3] M. Zhang, B. West, J. Lockwood, R. Indeck, M. Franklin, R. Cytron, and R. Chamberlain, "Reconfigurable Hardware-Based Database Searching," in preparation, 2002.

[4] Dan Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[5] S.F. Altschul, W. Gish, W. Miller, E.W Myers, and D.J. Lipman, "Basic Local Alignment Search Tool," *J. Mol. Biol.*, **215**:403-410.