An Architecture for Fast Processing of Large Unstructured Data Sets

Mark Franklin*†, Roger Chamberlain*†, Michael Henrichs†,
Berkley Shands*, and Jason White†
*Dept. of Computer Science and Engineering, Washington University in St. Louis
†Data Search Systems, Inc., St. Louis, MO

Abstract

This paper presents a general system architecture tailored to performing searching, filtering, compression, encryption, and other operations on unstructured data streaming from a disk system. The system achieves high performance on such applications by providing for parallelism, hardware-application specialization and reconfiguration, and hardware placement near the disk systems. A limited prototype of a single compute node has been implemented and is described. The prototype is tailored to applications involving complex searching and its performance is compared to a pure software implementation having the same search capabilities. Performance is considered in terms of data set size, query string hit rate and query complexity. Performance results as a function of these parameters are presented and the results indicate that, for data set sizes above 1.4 MB, the prototype compute node is between one and two orders of magnitude faster than a pure software implementation. At high data set sizes, on an individual node, speedups of about 200 and a sustained throughput of 300 MB/sec have been achieved.

1 Introduction

Over the past thirty-five years both semiconductor and magnetic technologies have advanced rapidy resulting in large increases in the bit densities available in both technology domains. In the semiconductor domain this increase has been expressed (in part) with the well publicized Moore's Law. The results have been dramatic increases in the number of transistors, memory and logic gates available on a chip, and a large decrease in the costs associated with processors. A similar even more dramatic increase in magnetic bit densities has resulted in a dramatic decrease in the costs associated with disk-based mass storage.

Starting in the early 1990s, magnetic storage densities have been increasing at a significantly faster rate
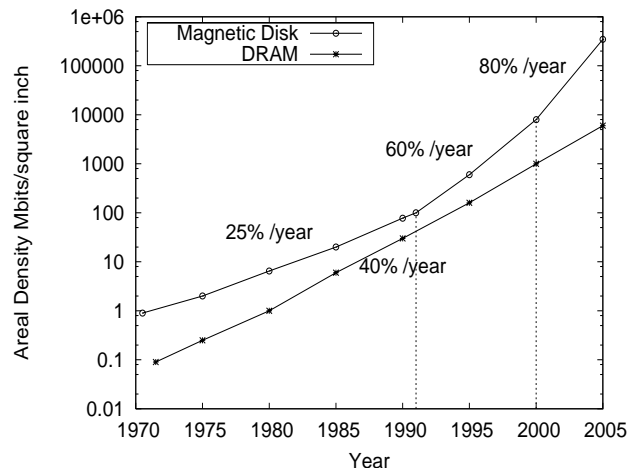

Figure 1: Semiconductor and magnetic bit densities

than semiconductor densities (see Figure 1). Over the past several years, the rate has increased further to more than twice that of semiconductor densities, and the costs of disk-based mass storage have plummeted.

This, along with other factors discussed below, has led to a large and growing gap between the amount of information being stored and our abilities to retrieve and process that information effectively. While this gap has been pointed out by others [1, 2], we consider its importance for the special domain of large unstructured data sets that are present in both the commercial and scientific communities. In the scientific disciplines of physics (e.g., astrophysics, astronomy & particle physics), biology (e.g., genomics & computational biology), and environmental science (e.g., satellite remote earth sensing) very large data sets, often in the tens of terabytes range, are being established that cannot be processed effectively with today's computer systems architectures. In the commercial environment, the growth of stored data sets (e.g., emails, documents, etc.), the legal requirements for preserving information,
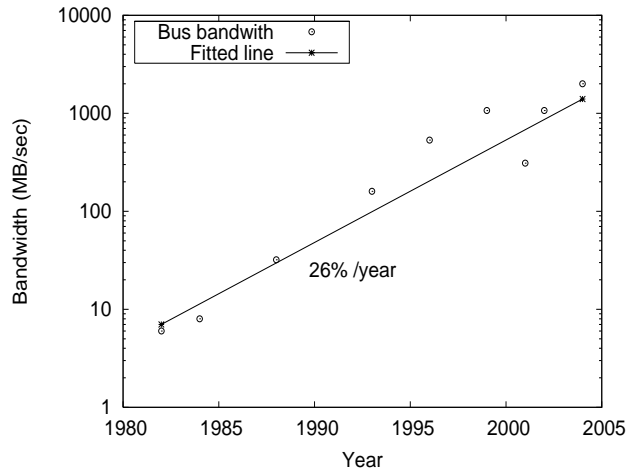
Figure 2: Disk bus bandwidth

2 Background and Motivation

2.1 The Storage-Processing Gap

The growing gap between the size of data sets and our ability to effectively process this data has its origin in a number of technological advances:

• **Availability of Inexpensive Mass Storage:** This is a result of the rapid advances in magnetic technology and the corresponding rapid decrease in the cost per bit of magnetic storage.

• **Availability of Inexpensive Sensors:** In scientific domains, new sensor and associated systems technologies (e.g., wide area sensor arrays) have significantly reduced the cost of gathering data, and the corresponding data set sizes have increased.

• **New and Broader Scientific Research Modalities:** Scientific disciplines have been both broadening their fields of inquiry to encompass larger physical domains while at the same time seeking to gather more data at higher resolutions and across additional dimensions (e.g., not just statistical means, but distributions). Additionally, entirely new fields are generating enormous data sets (e.g., genomics). Associated with this is the trend towards developing more comprehensive computer models that require large data sets for parameterization and verification.

• **The Computer System Bottleneck:**
A− The Processor Bottleneck: In recent years, single processor performance has not tracked the growing requirements of large data sets. In both the scientific and commercial domains, more complex model and processing requirements have led to an increase in the number of operations that must be performed per byte of data retrieved from the disk.
B− The Bus I/O Bottleneck: The interconnect bandwidth between the disk system and the processor has not kept pace with either the increase in disk capacities or the performance of processors. This trend is shown in Figure 2 where a best fit line associated with typical bus system bandwidths (e.g., PC, XT, ISA, PCI, PCI-X, PCI-Express, Fibre Channel, IDE/Ultra ATA, SATA, etc.) is shown. The growth rate in bandwidth is under 30% per year, significantly lower than that of either processor performance or disk densities. This, in conjunction with increased storage capacities and requirements, has resulted in the I/O bus often being a significant performance bottleneck. This bottleneck is often less of a problem with structured databases, where disk seek times may limit effective data rates. However, this is not the case with many scientific data sets and, increasingly, with many commercial and security/intelligence derived data sets.

and the increased use (e.g., mining) of large databases to aid in planning and marketing has led to data storage systems and associated processing being a key systems performance bottleneck. Unless new paradigms for computer systems architecture and associated applications implementation are developed, this trend is likely to continue [3].

This paper presents a new computer architecture that has been tailored to deal with this problem. The design goal is to achieve a performance improvement two orders of magnitude over current systems. The proposed architecture achieves higher performance by employing a set of techniques including: (a) exploiting access and processing parallelism at the level of the disks and disk groups, (b) using processor and FPGA-based pipelines to speed computation on data streaming from the disk, (c) utilizing hardware specialization and reconfiguration (via FPGAs) to provide fast computational capabilities, and (d) providing for the tailoring of the architecture (e.g., number of processors in a pipeline, tasks associated with the pipeline stages and reconfigurable logic) to the specific application being considered.

The remainder of this paper is divided into four sections. Section 2 expands on the needs and motivation for this new architecture and reviews other efforts in this area. Section 3 presents the architecture, an implemented prototype and some selected performance measurements. Section 4 considers a set of experiments that demonstrate the effectiveness of the approach when operating on fast complex search queries. The final section summarizes the work and discusses our future plans in this area.

C– Data Access Patterns: Unstructured data is typically generated and stored sequentially in long records and processing is performed across streaming data. This often results in data streaming off the disk at the disk rotational speeds. This is in contrast to relational and structured databases that have many relatively short data records with multiple distinct fields where accesses are often limited by seek times. However, the architecture proposed is also appropriate when large RAID-based disk techniques are employed and result effectively in high speed streaming data from the storage system.

2.2   Prior and Related Work

Work concerning methods for improving the access, processing, and performance of large databases has a long history. A large body of research and development work occurred during the 1970s and 1980s. These efforts were generally unsuccessful and the approaches taken were largely abandoned. As the problems with very large data sets increased, new efforts began in the mid 1990s and continue to this day. These efforts are briefly reviewed below.

Early Database Machines
In the 1970s and early 1980s, a good deal of research was done on "Database Machines"[4, 5, 6, 7]. Generally these machines provided special purpose scan, search, or associative memory capabilities oriented towards structured databases. Some machines had provisions for detecting and operating on individual records and record sub-fields. The focus of this work was on commercial applications whose operations were limited both in functionality (e.g., matching) and in the data types considered (e.g., text). The hardware developed was often coupled to one or more lower level disk objects (e.g., the individual disk head, track or entire disk). While some database machines demonstrated significant performance improvement on selected applications, they did not achieve long term success. The reasons for this include the high cost of the special purpose hardware and (at that time) its limited use in the database/disk environment. Additionally, the hardware operations tended to be less helpful when dealing with complex queries associated with the newer query languages and associated complex database applications.

Intelligent/Active Disk Systems
Over the past ten years there has been a surge of research in new types of database machines referred to as "Active Disks" (also "Intelligent" or "Smart" disks). There are several groups that have worked or are working in this area (Univ. of Cal. at Berkeley, Univ. of Cal. at Santa Barbara, Univ. of Maryland, Carnegie-Mellon Univ., Northwestern Univ.) [1, 8, 9, 10, 11, 12] with some overlap in their activities. In most cases, the idea is to place one or more commodity processors at the level of the disk and have these processors undertake a variety of disk operations (e.g., search, sort, group by, etc.). Operation results are fed back to the main processor. Thus, certain basic disk operations are off-loaded to the disk processors. Since these processors are fast, simple, commodity processors with standard I/O interfaces, overall application performance may be improved at relatively low cost. Additionally, since the processors are intended to be bundled together with the disks, as the size of a disk farm increases, the associated disk processing power scales directly.

This research has been successful in developing an important approach to dealing with large databases. Performance gains of greater than 50 to 100% on selected applications have been reported (with small systems) with the performance scaling as the number of disk processors employed increases. These results are principally based on the development, integration and use of specialized simulation models. Our architecture builds on this work but seeks to achieve significantly higher performance through use of additional levels of parallelism and functional specialization. Additionally, unlike much of this prior work, our focus is principally on unstructured data sets.

COTS (Commercial Off-The-Shelf) **Clusters**
A popular and cost-effective approach to dealing with large data sets is to use clusters of off-the-shelf computers (e.g., Linux Clusters) that are networked together. Separate disk(s) are associated with each of the computers, and the database is distributed across the disks. This architecture achieves many of the objectives associated with active disks. Uysal et al. [9] have shown (using simulation models) that over a benchmark of eight applications, when sufficient cluster bandwidth is available, there is little difference between the performance of such clusters and a proposed active disk system. We are aiming at achieving significantly greater computational performance by having increased parallelism and hardware specialization available at disk and disk group levels.

3   Architecture

The hardware architecture being developed will employ arrays of parallel compute engines containing reconfigurable components. The overall system is illustrated in Figure 3.
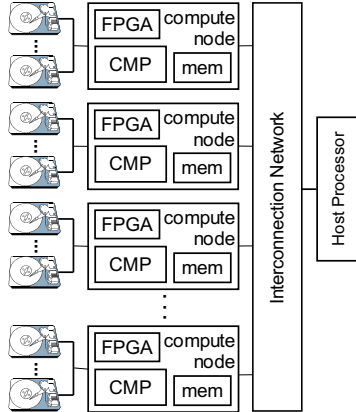
Figure 3: System architecture

Compute nodes are constructed with Field Programmable Gate Arrays (FPGAs) and chip-multiprocessors (CMPs). Currently available CMPs include network processors from Intel as well as the Xilinx Virtex-II Pro series. These compute nodes enable both functional specialization and pipelining of computational resources. Each node is assigned to a disk subsystem (comprising one or more drives) and operates in parallel on data streaming from the disks.

The block diagram of an operational prototype compute node is shown in Figure 4. In the prototype[1], the processor is a 2.4 GHz Intel Xeon with 2 GB of memory and the reconfigurable logic is a Xilinx Virtex-II FPGA. The disk subsystem is constructed using a set of 15,000 RPM Ultra320 SCSI drives organized in a RAID-0 configuration. On this prototype, we have demonstrated sustained read performance of greater than 800 MB/sec for continuous 500 GB file reads. We have also demonstrated sustained data throughput of greater than 400 MB/sec from the disk array into the FPGA.
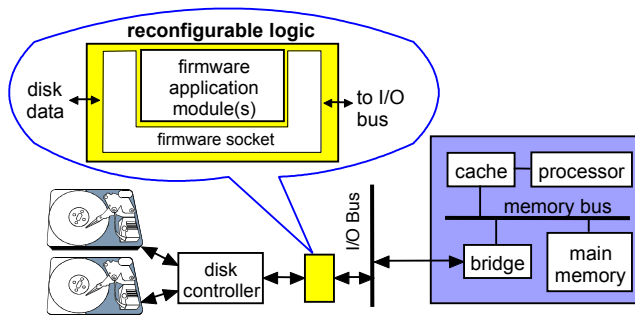

Figure 4: Prototype node architecture

With this configuration, various tasks can be performed on data streaming from the disk including such items as data filtering, both exact and approximate searching, compression, encryption, etc. For example, we have previously demonstrated [13] an operational Smith-Waterman algorithm [14] that is used extensively in computational genomics. In [15] we describe the deployment of the BLAST [16] biosequence search application on this system.

Deploying an application on the FPGA requires three steps: (1) designing the reconfigurable hardware component (e.g., using Verilog or VHDL); (2) loading the configuration information (i.e., bitfile) onto the FPGA; and (3) performing any required initialization and/or startup required by the design. We address the first step by making available a set of library designs (e.g., those listed in the above paragraph) for use by application developers, limiting the need for hardware-level design to take place for each new application. Using best practices for configuration loading and initialization [17], the second and third steps can be accomplished quickly (10s of ms). This time is comparable to seek times and rotational latencies associated with the disk subsystem.

A key element in effectively utilizing the above architecture is in partitioning the algorithm across the available FPGA and processor resources. A thorough understanding of the application and algorithm generally results in an implementation where the most time consuming portions of the algorithm are placed in the FPGA (sometimes in a pipelined fashion), while less time-critical portions are placed in the processor. Additionally, the proper placement of temporary data within the available memories is important. For the applications cited above, we have been able to arrive at implementations that can keep up with the data as it streams off the disk.

A key application that is considered in the remainder of this paper concerns the problem of performing complex approximate searching on large data sets. Say that we have a set of N_s text words or byte strings (e.g., "Bush", "Baseball", "Blair", "Soccer"), and we are interested in finding where logical combinations of these strings are present in the data store. For example, say that we are interested in scanning all English speaking newspapers published in a given day (assume these have been stored on our disk drives) that contain a general logical expression over the set of $N_s = 4$ words given above. One possible query is as follows:

```
((Bush NEAR[200] Baseball) AND
    (Blair NEAR[200] Soccer))
```

This query expresses the following conditions: (1)

the string "Bush" is found within 200 characters of the string "Baseball"; (2) the string "Blair" is found within 200 characters of the string "Soccer"; and (3) both conditions (1) AND (2) hold. Note that in this expression, the number of logical operators, N_l, is equal to 3 (AND is included once, NEAR is included twice). The current set of combining operators supported include AND, OR, NOT, NEAR, and ANDTHEN. The operators AND, OR, and NOT perform their traditional Boolean logic functions at the file level. The operator NEAR is equivalent to AND with the additional constraint that the matching strings must be within a given distance of one another in the file (default distance = 10 characters). The operator ANDTHEN is equivalent to NEAR with the additional constraint that the first term must occur earlier in the file than the second term (i.e., x ANDTHEN y imposes a precedence constraint on x and y). For example:

```
((Bush ANDTHEN[200] Baseball) AND
      (Blair ANDTHEN[200] Soccer))
```

requires that "Bush" preceed "Baseball" and that "Blair" preceed "Soccer", retaining the 200 character distance constraint. Note that in the current implementation, the individual strings are limited to 32 characters.

An important capability built into the system relates to approximate matching. In many instances, information is corrupted due to noise, to misspellings, or to misdirection. Additionally, sometimes characters are capitalized and sometimes not. Our system supports approximate matches of three forms. First, character matches can be case insensitive; second, individual characters can be designated as wildcards in which case any database character will be a match; and third, a count, k, of allowed mismatched characters can be present. Thus, up to k characters in the string can be wrong and the string will still be considered a match.

Note that the logical expression may be represented as a tree with the strings located at the leaves of the tree and the logical operators located at the interior nodes. Given this view, a simple partitioning of the algorithm leads to the FPGA performing the searching/matching operation for each of the tree leaves (under k-misses). On finding a hit (i.e., a match between the query word and a string obtained from the data store), the FPGA reports this information (along with its position in the file) to the processor. The processor then performs the logical operations and determines if the overall expression is valid. Implementation details of this FPGA design are described in [18].

In addition to the three parameters, N_s, N_l, and k, the data set size, F_n, and the string hit rate (i.e., number of matches on the individual words normalized by the file size), H_s, together can be viewed as indicators of overall query complexity. Naturally, one would expect as complexity increases performance decreases, as measured by an increase in the time required to service a query. The next section considers the performance of this system for different levels of complexity and compares its performance to that of a software implementation.

4   Performance Evaluation

The approximate searching described earlier was implemented and a series of experiments were undertaken to quantify performance. The objective was to compare performance of the FPGA/processor system with a pure software implementation over a set of queries that spanned significant portions of the query complexity space.

A set of sixteen Ultra320 drives were used to store unstructured data sets of varying sizes. Table 1 below indicates the range of experiments that were conducted.

Parameter	Range
Data set size, F	0.5 MB - 1.8 GB
Num. strings/query, N_s	1 - 4
Query string size, S	6 - 28 char
Num. logical operators, N_l	0 - 3
Num. misses/string, k	0 - 10
String hit rate, H_s	6 - 400 hits/MB

Table 1: Query and data set characteristics

The data itself consisted of synthetically generated text designed to provide the ability for controlled experimentation. A series of experiments were performed over two query sets, a low complexity set and a high complexity set. Low and high complexity corresponds to the left and right entries in the above table for N_s, S, N_l, and k. Each experiment consisted of the execution of ten searches for each data set size. Each point on the plots represents an individual search run.

The results of performing these searches are shown in Figures 5 to 8. Figures 5 and 6 show the time required for searches over a range of data set sizes for low and high complexity queries. Each of these searches is with a relatively low hit rate (under 100 hits/MB). As expected, the time increases as the data set size increases, and also increases as we move from low to high complexity queries. For example, in general for the software-based system, high complexity searches are an order of magnitude more time consuming than low complexity searches. For the FPGA-based implementation, while
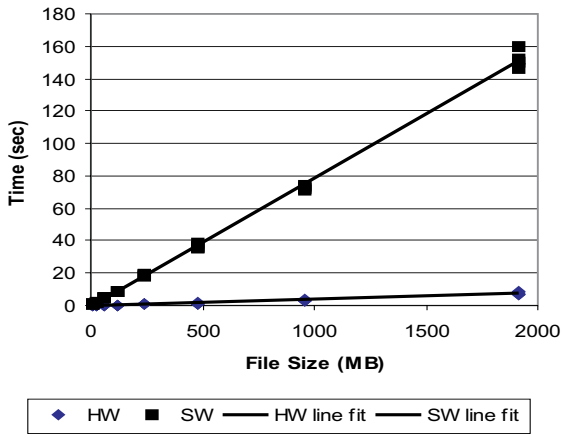
Figure 5: Search time vs. data set size for low complexity queries

Data Set Size	Low Complexity Time(sec)	High Complexity Time(sec)
60 MB		
Software	4.5	48.5
FPGA	0.25	0.25
960 MB		
Software	72.8	823
FPGA	3.1	3.2

Table 2: Search times for different query and data set sizes

the time increases with data set size and complexity, the increase is small since the major task of string matching occurs within the FPGA at the same rate that the disk provides the data. Only the relatively small computation associated with performing the logical functions is done in software.
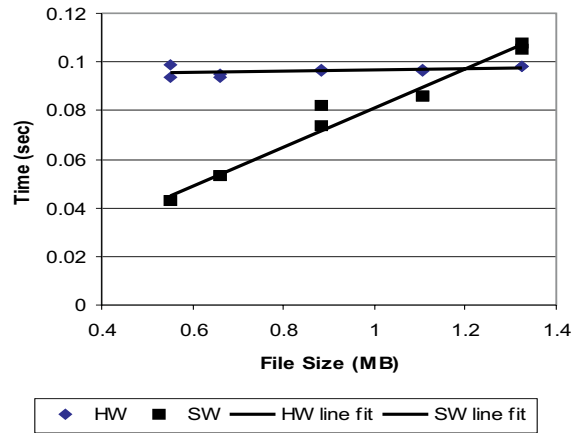
Figure 7: Search time vs. small data set size for low complexity queries
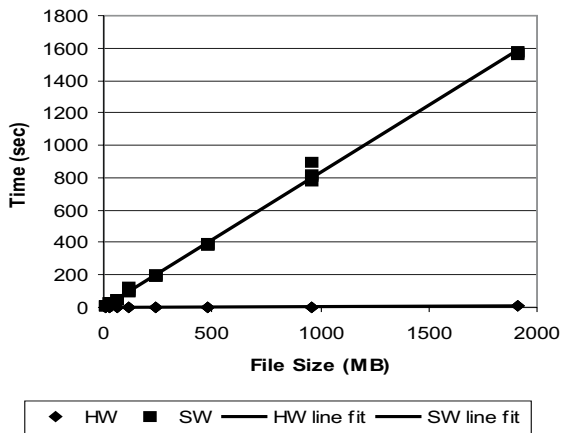

Figure 6: Search time vs. data set size for high complexity queries

These figures and Table 2 also show the enormous gain associated with this design over a strictly software implementation. For low complexity queries on data sets above about 1.4 MB the FPGA system is about an order of magnitude faster than the software system. For high complexity systems the gain is even more impressive and is over two orders of magnitude.

As the data set size becomes very small, the pure soft-

ware approach does better than the FPGA approach. This is shown in Figure 7, which expands the small file region of the low complexity plot of Figure 5. Below a data set size of 1.2 MB, the pure software approach is faster than the FPGA approach. In this region, there is relatively little gain associated with the hardware matching when compared with the logical operations. Additionally, there is some extra overhead associated with movement of data between the FPGA and the processor. Thus, for sufficiently small data sets we can expect a pure software approach to be superior.

One final issue of note relates to the effects of hit rate on the relative performance of the two approaches. This is shown in Figure 8 for a low complexity set of queries operating on a relatively small file (100 MB). Each point on this plot is the mean of four runs. For relatively low hit rates, the order of magnitude gain from using the FPGA-based system is shown. As the hit rate increases, both the pure software and FPGA-based system take more time. With more hits, the logical func-

tions, implemented in software on the FPGA-based system, effect the performance in the same manner as the pure software system. Thus, with increasing hit rates, the percentage performance difference between the two systems narrows.
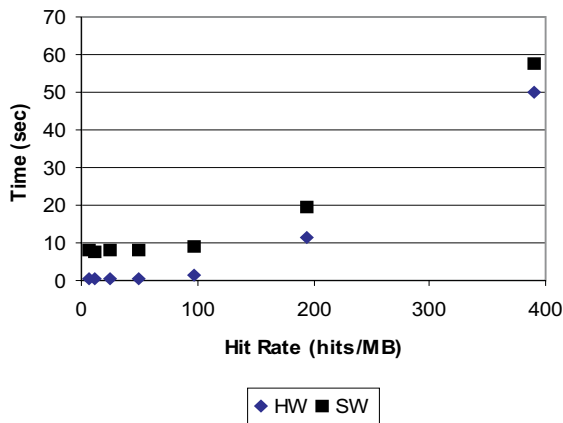

Figure 8: Search time vs. hit rate

5   Summary and Conclusions

In this paper we have presented a general system architecture tailored to performing searching, filtering, compression, encryption, and other operations on unstructured data streaming from a disk system. The system achieves high performance on such applications by providing for parallelism, hardware-application specialization and hardware placement near the disk systems.

A limited prototype of a single compute node has been implemented and an approximate text search application has been deployed on it. Performance of this system has been compared with that of a pure software implementation of the same search capabilities. Performance results as a function of data set size, query complexity and string hit rates have been presented. These results indicate that for data set sizes above 1.4 MB, the FPGA-based system is between one and two orders of magnitude faster than a pure software implementation. At high data set sizes, on a system containing a single compute node, speedups of about 200 and a sustained throughput of 300 MB/sec have been achieved. As the size of the data set increases along with the number of disks, multiple compute nodes can be employed in parallel and, for this application, performance will increase proportionately.

While the general system architecture is an effective one for the application class targeted, the node architecture is rapidly evolving. For example, although the use of FPGAs for fast string matching is very effective, there are certain drawbacks. In the future we will require the ability to dynamically change the FPGA structure in response to incoming queries and application changes. Unless the FPGA is sufficiently large to hold multiple functions, downloading new configurations is currently time consuming and does not provide suitable responsiveness in a dynamic system environment. Additionally, certain applications are better implemented on parallel or pipelined sets of general purpose processing engines. Chip multiprocessors (CMPs) are now available in a number of forms. Network processors having sixteen or more compute engines are commercially available. Intel has also recently announced that they will be providing (as a first step) dual processor Pentium-like chips in the future. Our future research will be focused on developing node level architectures that combine the use of CMP-based pipelines with FPGA-based coprocessors. Such systems will combine high performance, fast reconfigurability, and relative ease of programming.

Acknowledgements

References

[1] E. Riedel, C. Faloutsos, G. Gibson, and D. Nagle, "Active disks for large-scale data processing," *IEEE Computer*, pp. 68–74, June 2001.

[2] J. Gray, "What happens when processing, storage, and bandwidth are free and infinite?," 1997. "http://www.research.microsoft.com/∼gray/talks-/IOPADS.ppt".

[3] N. B. R. A. P. on Cyberinfrastructure, "Revolutionizing science and engineering through cyberinfrastructure," tech. rep., National Science Foundation, CISE, Jan. 2003. www.cise.nsf.gov/sci/reports/.

[4] E. Ozkarahan, S. Schuster, and K. Smith, "Rap - an associative processor for data base managment," in *Proc. of International of AFIPS, National Computer Conf. 44*, 1975.

[5] D. Smith and J. Smith, "Relational database machines," *IEEE Computer*, Mar. 1979.

[6] H. Boral and D. DeWitt, "Database machines: An idea whose time has passed?," in *Proc. of International Symp. on Database Machines*, Sept. 1983.

[7] J. Dixon *et al.*, "Associative file processing method and apparatus," *US Patent # 4,464,718*, Aug. 1984.

[8] K. Keeton, D. Patterson, and J. Hellerstein, "A case for intelligent disks (IDISKS)," in *Proc. of Sigmod Record*, Sept. 1998.

[9] M. Uysal, A. Acharya, and J. Salz, "Evaluation of active disks for decision support databases," in *Proc. of Symp. High Performance Computer Architecture, HPCA-6*, pp. 337–348, Jan. 1999.

[10] A. Acharya, M. Uysal, and J. Salz, "Active disks: programming model, algorithms and evaluation," in *Proc. of ASPLOS-VIII*, pp. 81–91, Oct. 1998.

[11] E. Riedel, "Active disks - remote execution for network-attached storage," tech. rep., Carnegie-Mellon University, 1999. Doctoral Disseration, Computer Science Tech. Rpt.-CMU-CS-99-177.

[12] E. Riedel, C. Faloutsos, and D. Nagle, "Active disk architecture for databases," tech. rep., Carnegie-Mellon University, Apr. 2000. Computer Science Tech. Rpt.-CMU-CS-00-145.

[13] B. West, R. D. Chamberlain, R. S. Indeck, and Q. Zhang, "An FPGA-based search engine for unstructured database," in *Proc. of 2nd Workshop on Application Specific Processors*, pp. 25–32, Dec. 2003.

[14] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 141, no. 1, pp. 195–197, 1981.

[15] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, and J. Lancaster, "Biosequence similarity search on the *Mercury* system," in *Proc. of IEEE Int'l Conf. of Application-specific Systems, Architecture, and Processors*, 2004. To appear.

[16] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, pp. 3389–402, 1997.

[17] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, pp. 171–210, June 2002.

[18] Q. Zhang, R. D. Chamberlain, R. S. Indeck, B. West, and J. White, "Massively parallel data mining using reconfigurable hardware: Approximate string matching," in *Proc. of Workshop on Massively Parallel Processing*, Apr. 2004.