GREO : A Commercial Database Processor Based on A Pipelined Hardware Sorter

Shinya Fushimi
Information Systems Laboratory
Mitsubishi Electric Co.
5-1-1 Ofuna, Kamakura, Japan

Masaru Kitsuregawa
Institute of Industrial Science
The University of Tokyo
7-22-1 Roppongi, Tokyo, Japan

1. Introduction

GREO is an attached database processor developed for midrange business computers. Since 1989, thousands of GREO's have been shipped in the Japanese midrange market.

GREO is the first commercial product of a *high-speed sorting hardware*, or *a hardware sorter*. GREO implements the $O(n)$ time pipelined merge sort algorithm with several enhancements[1]. To fully utilize this sorting capability, the language processor for GREO compiles a given query into a sorting-oriented dataflow graph, which is in turn executed by the hardware sorter in conjunction with *multi-microprocessors*.

Another unique aspect of GREO is that it is a commercial product developed in collaboration between a university and an industry (which are actually authors' current affiliations). In spite of not a few "academic flavors" in its architecture, it succeeded in faithfully meeting practical customers requirements, i.e., improving performance of a variety of *existing* (relational and even *non-relational)* business applications.

The paper describes both of its technical and practical aspects. First, the query processing model underlying GREO architecture is presented. Its architectural features are then described. Our experiences on making research results an actual commercial product are also reported.

2. Query Processing Overview

The query processing strategy is based on *data stream oriented processing* [2], which is also called *pipelined parallelism* in [3]. That is, a given query is first decomposed into a set of primitive operations, which are then organized to form a *query processing pipeline*. Data read from disks are directed to this pipeline, where each of primitives are applied to the data stream in parallel.

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0449...$1.50

Such primitives are regarded as stages of the pipe, hence called *filters*. For example, the following SQL query:

> SELECT C1, C2 FROM T WHERE C1 < 100
> ORDER BY C2
>
> is compiled to a three stage pipe:
>
> stage 1 filter: project fields onto C1, C2
> stage 2 filter: select records where C1<100
> stage 3 filter: sort records by the key:C2.

We can further consider complex queries referencing multiple files (e.g., joins), or comprising arithmetic operations (e.g., aggregations). Thus a query is generally organized as a tree, or more generally, a *graph*, of filters. Multiple number of data streams are activated and poured along this structure. A *dataflow control* is employed to govern the overall execution of a query.

The ideal implementation of this model pursues $O(n)$ time, or *on-the-fly*, execution of all kinds of filters. Because every filter can run on-the-fly if the data are *sorted*, $O(n)$ time hardware sorter turns out to be the key component there.

3. Hardware Architecture

3.1. Structure of GREO Hardware

GREO hardware consists of a *hardware sorter* and a *data stream processor*(Fig.1). The hardware sorter is devoted to sorting (a *sorting filter*), while the data stream processor is responsible for executing any other primitives, such as selections, projections, joins, and arithmetic computations (*non-sorting filters*).

Typically, all of records are first directed to the data stream processor, where filters for selections or projections are applied on-the-fly. The resultant records are sent to the sorter. The sorted recoreds are again directed to the data stream processor, and other types of on-the-fly filters (e.g., aggregations, joins) are performed. The final results are sent back from the data stream processor. By combining appropriate filters, GREO provides various types of database functions. Since it executes all of filters in parallel, it can be considered a dynamically reconfigurable, multi-stage subsegment of the query processing pipeline. A complex

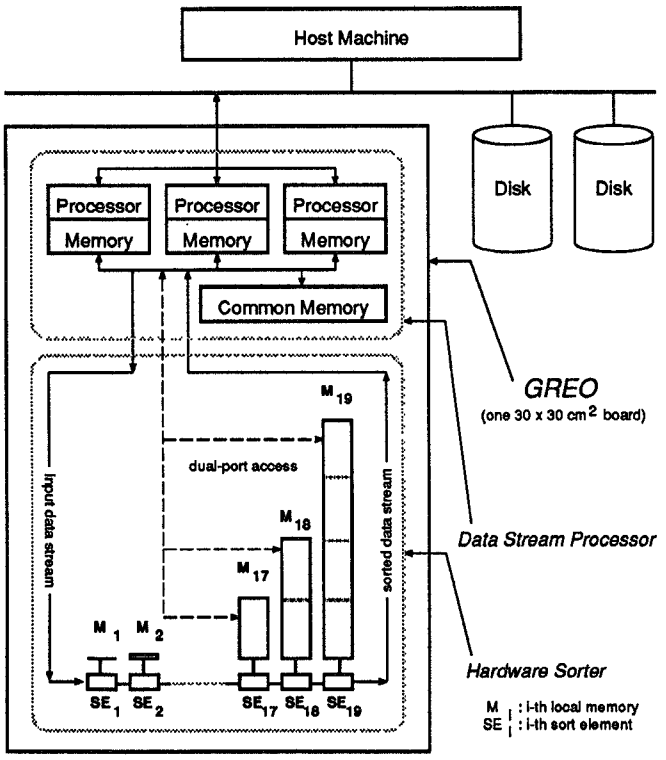

Figure 1: Hardware Organization

dataflow graph is executed by cascading these subsegments with multiple GREO's.

3.2. Hardware Sorter

The hardware sorter implements the *two-way pipelined merge sort algorithm*[1][4][5]. To sort 2^n records, n sorting elements $\{SE_1, \ldots, SE_n\}$ are prepared, and connected in a pipeline fashion. SE_i associates itself with a local memory of 2^{i-1} records in size. A set of records are directed to SE_1 one by one, and sorted records are continuously output from SE_n.

The brief sketch of the algorithm is given here. Each of elements repeats to input two sorted strings, merge them into one sorted string, and output it to the next. By overlapping these three stages as much as possible, we have an algorithmic description of the $O(n)$ time pipelined merge sort (Fig.2). For example, the first sort element SE_1 repeatedly inputs a pair of records, merges them, and outputs a sorted string of 2 records in length. To do that, it inputs the first record (key:8) onto its own local memory. Next, it inputs the second one (key:2) and merges them. Since $8 > 2$, the record with key:8 is then output to the next element SE_2. At the same time, the first record in the next pair (record with key:1) is input and stored. In the next cycle, the remaining record (key:2) is output while the next pair is formed and merged (key:1 and key:3). SE_1 continues this process till records are consumed. Other elements

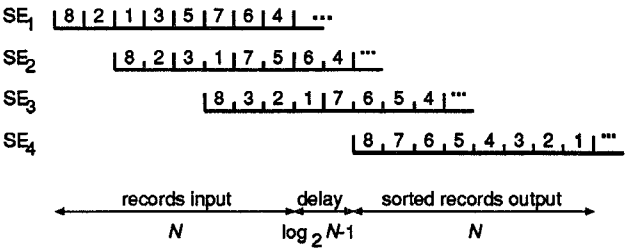

Figure 2: Pipelined Merge Sorting

SE_i $(i = 2, \ldots, n)$ execute the same procedure except that the length of strings are doubled as i increases.

The current GREO implementation incorporates 19 sorting elements in 10 LSI chips with 8MB local memory capacity in total. When data to be sorted is larger than its total memory capacity, we first direct entire data to the sorter. The sorter generates a sequence of 8MB sorted strings. We then merge these strings by the data stream processor. To offer a buffer area enough for merge operations, the local memory banks of the sorter are designed to have dual-port access capability. Also, the sorting element is designed to dynamically tune themselves for variable length and number of records[5].

3.3. Data Stream Processor

The data stream processor realizes all of non-sorting operations. It allows parallel executions of multiple non-sorting filters.

The data stream processor is just a pool of microprocessors in reality. To achieve high performance enough to keep up with the flow of data (i.e., data transfer rate of disks) at reasonable cost, and to make it flexible enough to implement a variety of database requests (exception handling in Japanese text processing, deeply-nested predicates, etc.), multiple general-purpose microprocessors are highly preferable. All of non-sorting filters are implemented by means of parallel/concurrent processes on these processors. Even the sorter is encapsulated inside some of these processes. Upon request, these processors are dynamically divided and allocated to each of filters (see the later section).

Although the logical design allows any number of any type of processors, current GREO incorporates three MC68020's as the data stream processor.

4. Software Architecture

4.1. Structure of GREO Software

The layered hierarchy of the software architecture is shown in Fig.3.

On the host machine to which GREO is attached, we have four layers. The top layer consists of applications. Just below the applications layer comes the

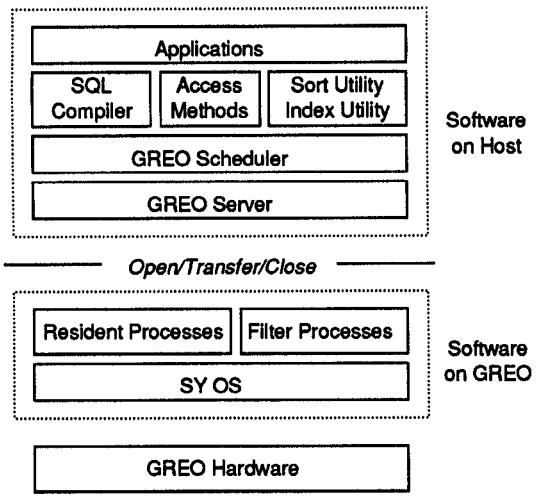

Figure 3: Software Organization

so-called middle-ware layer (e.g., SQL processor, the 4-th generation languages, access methods, sorting utility, etc.) that compiles queries to hook database access requests worthy of directing to GREO. When the OS running on the host machine detects that GREO is attached, such database requests are sent to the *GREO scheduler*. The GREO scheduler is the resource manager for (possibly multiple) GREO('s) attached to the machine; it assigns the required number of GREO's to the request and activates the specialized server process, or the *GREO server*. The GREO server is responsible for executing the database request by running GREO's.

On GREO, we have two layers of software. The multiprocessor/multiprocess kernel operating system called SY underlies any database processing. SY runs on each of microprocessors, and provides both of message-based, and lock-based interprocessor/interprocess communication primitives as well as process and memory management. Several parallel processes are activated on it; some of these processes are made resident to implement I/O interface to the host machine. Other processes are dynamically created, and execute the specific database processing in their contexts. The latter set of processes actually implements the concept of filter, and is thus called *filter processes*.

4.2. Filter Implementation

Filter processes are software abstractions of GREO hardware. Basically, one filter process corresponds to one database primitive. For example, the selection filter is in charge of sieving out undesired records from a data stream. The sorting filter process is devoted to sorting by utilizing the hardware sorter. To reduce overhead in interprocess communication between these filters, however, major filters are integrated into a complex one (e.g., a selection filter with projection).

When the GREO server starts to execute a request, it first identifies a set of necessary filters, and issues *open* instruction to GREO to activate them. GREO server then starts data transfer from disks to these processes by repeating *transfer* instructions, while the filter processes run against this data stream in pipeline fashion. The execution is finished by the *close* instruction. This abstract level interface (*open/transfer/close*) does not depend on filters semantics, thus makes GREO highly flexible and extensible.

Filter processes are also units of SY's processor allocation. They are coded so that any number of parallel instances can run consistently. When *open* is issued, for each of designated filters, SY allocates the appropriate number of processors, and starts identical filter processes on them.

5. Making It a Commercial Product

In [3], it is suggested that, in contrast with the architecture based on a specialized hardware, shared-nothing multiprocessor architecture by using general-purpose microprocessors are preferable. Since the hardware sorter, a typical example of a specialized hardware, is one of major components of GREO, our experiences and customers evaluation on it may be informative to database researchers.

5.1. Historical Background

The hardware sorter was originally a part of the former database machine project at the University of Tokyo[6], [2]. Our initial motivation to GREO project was to make this sorter an actual product in some computer market. We placed a target on midrange business computers, since the most drastic performance improvement was expected there.

When we started the GREO project in 1987, we first conducted market analysis in Japan. The result was rather different from those which we had imagined at the university.

The first observation in the analysis was that the compatibility of existing business applications was of great importance; even in the midrange market, hundreds of applications had been used on a machine. Users had already invested not a little money to develop them, and could afford to spend no more extra money to the system. Even if we have had offered very high speed database machine, no users welcomed it if the machine required serious rewriting of their applications.

Also observed was that users requirements were focused on user interface, performance, hardware volume, and cost; no serious requirements on underlying technology (e.g., no adherence to UNIX). Even they did not care whether DBMS was relational or not.

Another point we observed in the Japanese midrange computer market was that batch-oriented pro-

cessing was not obsolete; users in this market did not always need an up-to-date image of data managed by on-line, real-time processing, but only wanted to complete transaction processing under rather relaxed time limitations (e.g., in one day). OLTP usually requires additional investments, hence they often preferred to batch processing at night. Since sorting is still a major bottleneck there, it turned out that the hardware sorter was much more effective than we had expected.

5.2. Product Design

According to these observations, we began hardware and software design. What we did first was to give up several implementation alternatives which had been considered practical reality in an academic world.

For example, because users require midrange computers to occupy less area, GREO should not be a back-end machine in a separate box, but be an attached processor consisting of only a couple of hardware boards. Also, electric power consumption is serverly limited. Under these physical limitations, we could not incorporate tens of processors, nor sorting elements with large storage. After several compromises, we reached a single $30 \times 30cm^2$ board implementation which integrated 3 MC68020's and 19 sorting elements with 8MB memory.

Also, we could not define new API's which allow programs to fully utilize GREO. This is due to severe applications compatibility; what to be accelerated was existing applications, not new programs written hereafter. Thus, GREO should be transparently accessed below the applications layer. This resulted in rewriting not a few middle-ware programs. Although some performance improvement was compromised, this perfect compatibility was most welcomed by users. Just after its debut, however, some customers began to request for programming interface to use GREO exhaustively. Instead of defining new API's for that, we developed the new optimizing SQL compiler specialized for GREO execution.

5.3. Evaluation

Although the ideal implementation (more number of state-of-the-art microprocessors, more number of sorting chips, more storage capacity, etc.) cannot be realized due to several practical limitations and requirements, we might say that GREO has been successful.

Customers reported us up to 100 times performance improvement of actual existing applications. This means that one-night batch processing can be completed in a few minutes. This scale of performance improvement was highly impressive to users, since this led them to direct cost savings, such as annihilating extra labor hours at night, or getting a chance to run new heavy applications such as strategic business data analysis. This performance figures also clarified that many users actually had batch-oriented applications where most of CPU time was consumed by sorting and its associated processings (selections, projections, joins, etc.).

6. Conclusion

In this paper, the query processing strategy and architectural features of GREO are described. Several aspects from practical viewpoints are also reported.

Summing up our experiences on GREO, GREO succeeded in providing impressive performance improvement with full compatibility at reasonable cost. Since batch-type applications are often preferred to in the Japanese midrange computer market, sorting is of great importance there. Customers evaluations are GREO is an actually cost-effective database processor.

To make it a commercial product, many compromises have been done. Much higher performance could be achieved if we ignore physical limitations by incorporating more powerful hardware. In commercial aspects, however, customers major concern should be respected.

We generally agree on the future directions in [3] that multiprocessor architecture comprising state-of-the-art general purpose microprocessors is most promising. Actually, the data stream processor of GREO follows this direction. We believe, however, some of the specialized hardware components will be still effective there. From our five year experiences, the hardware sorter will be undoubtedly one of such components. Another possible candidate may be found around the disk system [7].

References

[1] Kitsuregawa, M., Yang, W., and Fushimi, S.. Implementation of LSI Sort Chip for Bimodal Sort Memory. In *Proc. of VLSI-89*, pages 285–294, August 1989.

[2] Fushimi, S., Kitsuregawa, M., and Tanaka, H.. An Overview of the System Software of a Parallel Relational Database Machine GRACE. In *Proc. of 12-th Very Large Databases*, pages 209–219, August 1986

[3] DeWitt, D., and Gray, J.. Parallel Database Systems: The Future of High Performance Database Systems. *Comm. ACM*, 35(6):85–98, 1992

[4] Kitsuregawa, M., Yang, W., and Fushimi, S.. Evaluation of 18-stage Pipeline Hardware Sorter. In *Proc. of IWDM-89*, pages 20–33, June 1989.

[5] Kitsuregawa, M., Yang, W., Suzuki, T., and Takagi, M.. Design and Implementation of High-Speed Pipeline Merge Sorter with Run Length Tuning Mechanism. In *Proc. of IWDM-87*, pages 144–157, June 1987.

[6] Kitsuregawa, M., Tanaka, H., and Moto-oka, M.. Application of Hash to Database Machine and Its Architecture. *New Generation Computing*, 1(1):63–74, 1983.

[7] Kitsuregawa, M., Nakano, M., and Takagi, M.. Performance Evaluation of Functional Disk System. *Proc. of 7-th Int'l Conf. on Data Engineering*, pages 416–425, April 1991