INVITED PAPER *Special Issue on Super Chip for Intelligent Integrated Systems*

Overview of the Super Database Computer (SDC-I)

Masaru KITSUREGAWA†, *Member*, Weikang YANG†, Satoshi HIRANO†, Masanobu HARADA†, Minoru NAKAMURA†, Kazuhiro SUZUKI†, *Nonmembers*, Takayuki TAMURA† *and* Mikio TAKAGI†, *Members*

SUMMARY This paper presents an overview of the SDC-I (Super Database Computer I) developed at the University of Tokyo, Japan. The purpose of the project was to build a high performance SQL server which emphasizes query processing over transaction processing. Recently relational database systems tend to be used for heavy decision support queries, which include many join, aggregation, and order-by operations. At present high-end mainframes are used for these applications requiring several hours in some cases. While the system architecture for high traffic transaction processing systems is well established, that for adhoc query processing has not yet adequately understood. SDC-I proved that a parallel machine could attain significant performance improvements over a conventional sequential machine through the exploitation of the high degree of parallelism present in relational query processing. A unique bucket spreading parallel hash join algorithm is employed in SDC, which makes the system very robust in the presense of data skew and allows SDC to attain almost linear performance scalability. SDC adopts a hybrid parallel architecture, where globally it is a shared nothing architecture, that is, modules are connected through the multistage network, but each module itself is a symmetric multiprocessor system. Although most of the hardware elements use commodity microprocessors for improved performance to cost, only the interconnection network incorporates the special function to support our parallel relational algorithm. Data movement over the memory and the network, rather than computation, is heavy for I/O intensive database processing. A dedicated software system was carefully designed for efficient data movement. The implemented prototype consists of two modules. Its hardware and software organization is described. The performance monitoring tool was developed to visualize the system activities, which showed that SDC-I works very efficiently.

key words: *parallel machine, relational database, SQL, parallel algorithm*

1. Introduction

The database management system is one of the most important components of the modern computer systems. Recent widespread adoption of relational database systems, mainly due to its ease of use, has led to the creation of larger and larger databases, which has brought about the great demand for super relational database servers that have much higher performance than current ones. One of the key features of relational database systems is their employment of the non-procedural query language, SQL, through which users can issue adhoc queries very easily. This stimulates the use of the relational database for query processing, while traditional database management systems have been mainly adopted in high traffic transaction processing systems.

Commercial database applications can be classified roughly into two types: transaction processing (TP) and query processing (QP). Considerable research has been done on TP. Large online systems such as banking with ATMs, reservation systems, and stock marketing systems have been constructed. This technology, which is characterized by very high traffic rate and relatively small access to the database, has already matured. Recently, relational database systems tend to be used for query processing applications such as decision support, market analysis, sales trend analysis, and information mining. Since queries are issued to the system in an adhoc way, usually queries access the attributes which are not indexed. Query processing for statistical analysis scans very large amounts of data and takes a very long time to complete. It can take several hours, even days in certain cases. Large amounts of data generated by transaction processing is accumulated and this data tends to be used by query processing for statistical analysis. The standardization of the benchmarks reflects the prevalence of such applications. TPC-A, B, C are well known benchmarks for transaction processing, and at present frequently used as performance metrics of the machine. Currently TPC is working to establish yet another benchmark called TPC-D, which is targeted for decision support queries.

Much more powerful machines are required to support query processing, since it has to scan very large amount of data and has to do some computation over it. Current mainframe machines are not necessarily sufficient. There are two directions that can be taken to improve the performance of relational database processing: the special purpose processor approach and the parallel processor approach. Searching is one of the most fundamental operations. Special hardware to accelerate the search operation has been researched [3], [7], [19], [22] and developed as a product [9], [18], [23] which performs interpretation of the physical record structure, predicate evaluation and the extraction of

Manuscript received January 17, 1994.

† The authors are with the Institute of Industrial Science, The University of Tokyo, Tokyo, 106 Japan.

necessary fields from the records efficiently. If search processing is done by the CPU, all the data from the disk must be transferred to the CPU via the channel. The bandwidth of the channel is usually limited and is a very expensive resource. Intelligent disk controllers which incorporate the search logic are currently used in mainframe computers. It is reported that these devices can decrease the load of the channel dramatically and increase the system throughput [18]. This filter processor is very effective for the reduction of the data but is not so helpful for the heavy relational operators such as join.

The other key function frequently used in database processing is sorting. Most of the current commercial products use a sort library as a preprocessor for the relational operations such as join, aggregation and duplicate elimination. Sorting is also heavily used for report generation and index creation. Due to the heavy load caused by sorting, special hardware engines have been developed. IDP (Integrated Database Processor) [17] by Hitachi modifies the vector processing unit to support the merge operation. By using those vector units iteratively, it can produce a fully sorted record stream. Another special sort machine prepares $\log N$ comparators connected linearly with each dedicated memory bank [4], [20]. This can sort the record stream in $O(N)$ time, while the uniprocessor machines take $O(N \log N)$ time. Linear time sorting means that the file read out from the disk can be directly fed into this sort accelerator. High speed sorting plays a very important role under practical applications. Since the end of 80's, large computer manufacturers, especially Japanese companies, have developed special database accelerators. Since considerable investment has gone into the development of host machines, application specific functionality should be designed to integrate with these systems with minimal impact. The sorter is added as an extension to the host system. A hardware sorter can boost the performance of join operations, but it is a basically sequential architecture and cannot exploit a high degree of parallelism.

Recently the computer industry has seen a shift from the use of proprietary systems to the use of open systems. Downsizing stimulates the replacement of the large mainframe based centralized system with an inexpensive microprocessor based distributed system using open software. For transaction processing applications a multiprocessor system fits very well and can attain very high transactions per second, which is much higher than mainframe machines. If such general purpose parallel processor systems based on inexpensive microprocessors can be used for relational query processing, hopefully we can realize very powerful as well as scalable systems. Parallel database processing has been an active research area for the last ten years [1], [2], [5], [8], [15], [24].

So far we have done research on performance issues on relational database systems. We have developed the parallel relational algorithms, high speed hardware sorter and the functional disk system (FDS) [12], [13], [15], [16]. FDS was implemented to determine whether introducing database functionality into the disk controllers leads to improved database performance. Based on our previous results, we started the SDC project in 1988.

SDC-I is an experimental prototype to prove the viability of microprocessor based parallel query processing servers [6], [10], [14]. One of the most unique features is in its use of the bucket spreading hash join algorithm which is robust against data skew. With this algorithm, SDC can achieve high degrees of scalability. Extra hardware was introduced into the interconnection network to assist the algorithm. A dedicated operating system was constructed to support efficient data transfers among the modules. These transfers are essential for data intensive database processing. SDC-I consists of several modules interconnected by the network, where each module contains five MC68020 microprocessors, while SDC-II under development employs seven MC68040's. The I/O system is much more enhanced compared with current scientific parallel machines. SDC-I was designed to have very powerful query processing capabilities. This paper overviews several aspects of SDC-I: its parallel relational algorithms in Sect. 2, its architecture in Sect. 3, its software system in Sect. 4, and its performance monitoring tools in Sect. 5. Section 6 presents our conclusion.

2. Parallel Relational Algorithm

In order for a parallel system to work effectively, the target application must have substantial inherent parallelism which is easily exploitable. Fortunately, relational query processing applications contain high degrees of parallelism. This parallelism can be categorized into three levels. Several queries can be processed in parallel (inter-query parallelism). Several operations within a query can be processed in parallel (intra-query parallelism). A single relational operator can exploit data parallelism (intra-operator parallelism). Thus large amounts of parallelism can be exploited for relational query processing.

Then we need an efficient parallel algorithm which can exploit this parallelism. Usually very sophisticated sequential algorithms tend to be difficult to be parallelized. SDC employs a new parallel algorithm named "Bucket Spreading Hash Join" [14]. Conventional parallel hash join algorithms are very sensitive to data skew, because the buckets generated by the hash function are assigned to the processors statically. Thus the size of the hash tables varies among the processors. In some cases, certain proces-

sors' memories may overflow and the others' may not. If overflow occurs, performance deteriorates significantly. The total execution time is determined by the slowest module. Thus the conventional naive parallel hash join algorithm or the parallel hybrid hash join algorithm are very fragile under non-uniformly distributed data. Any parallel processing system must pay careful attention to load balancing in order to achieve linear scalability. The bucket spreading hash join algorithm is designed so that it works well even if the distribution of data is skewed appreciably.

The algorithms utilizes a shared nothing architecture and assumes that all the relations are fully declustered over the modules. The bucket spreading hybrid hash join algorithm works as follows. Let the smaller relation be R and the larger be S.

i) Build Phases: Each module applies the hash function to each tuple of its portion of the relation R, then send out the tuples over the interconnection network, named bucket flattening omega network. This functional network automatically distributes the tuples with the same hash id equally among the modules. If the size of the relation R is small enough, the whole relation fits into the main memory space. However, if the relation size exceeds the size of main memory, then the dynamic destaging mechanism is invoked [11], [21]. The largest bucket is selected and the tuples of that bucket are destaged into the disks. The memory space occupied by this bucket is released for use by incoming tuples. Everytime the memory space becomes full, dynamic destaging occurs. When SDC finishes reading all the tuples from the disks, it examines the size of the buckets in memory. Since the bucket flattening omega network distributes each bucket evenly over the modules, a certain coordinator can schedule the bucket assignments with its local information. Each module exchanges the bucket fragments among each other following the given schedule and builds its own hash table. This scheduling and data exchange seems to be an extra cost, since such processing is not required by the naive parallel GRACE hash join. However, this contributes to the avoidance of hash table overflow and can utilize the memory space with the highest degree of efficiency. In addition, the time necessary for it is small, since data exchange involves not the disk I/O but only communication over the interconnection networks, where the recent network bandwidth is much higher than that of the disks.

ii) Probe Phase: The relation S is read out from the disk and the hash function is applied to each tuple. If the hashed value falls into one of R's bucket ids which are stored in the current module's main memory, that tuple is probed against the local hash table and the result tuples are produced. The result tuples are again hashed over the attribute for the next operation and sent out over the bucket flattening network. If the hashed values falls into R's bucket ids which are stored on the current module's disk, the tuple is stored back into the disk. Otherwise, the tuple is sent to the corresponding module according to its hashed value. When all the tuples are finished being read from the disks, the result tuples for the buckets resident in memory are produced and the nonresident buckets are stored back to the disks. At this time, the statistical information on the buckets stored on the disks are available for scheduling the remaining buckets. The assignment of buckets to the processors is determined with this information, this process is called bucket tuning [11]. Once the schedule is fixed, build and probe phases are repeatedly performed until all the tuples have been processed.

3. Architecture of SDC-I

3.1 Global Architecture

As shown in Fig. 1, SDC-I globally employs the distributed memory architecture where modules are connected through the interconnection networks. Each module itself is a symmetric multiprocessor system (SMP). Although the single processor per module approach is much simpler, we adopted SMP as a unit of the system. This relies on our belief that in the near future a single chip will include multiple processors to increase the performance beyond the degree of superscaler parallelism. Also, we believe that the system software will support both shared memory and message passing paradigms.

A single module contains four MC68020 microprocessors. Processors are connected through two buses, one for high speed data transfer (H-bus) and the other for handling control information and mutual exclusion (C-bus). The H-bus is used solely for bulk data transfer. Memory is also composed into two portions: an 8 Mbytes data memory for the raw tuple data and a 2 Mbytes control memory for storing the control data structures such as page address lists, hash table entries, and memory consumption statistics for each staging buffer. Each module controls two disk drives in parallel. The disk controller identifies the tuple boundaries, generates the logical page and invokes DMA transfers to the data memory. Two disk controllers are managed by the control processor (CP), which manipulates the page table on the control memory and sets the DMA information for the disk controller. The control processor also manages all the activities of the module, such as the synchronization of four processors.

Modules are connected through two kinds of interconnection networks: the data network and the control network. The data network offers high-speed channels for data transfers, while the control network

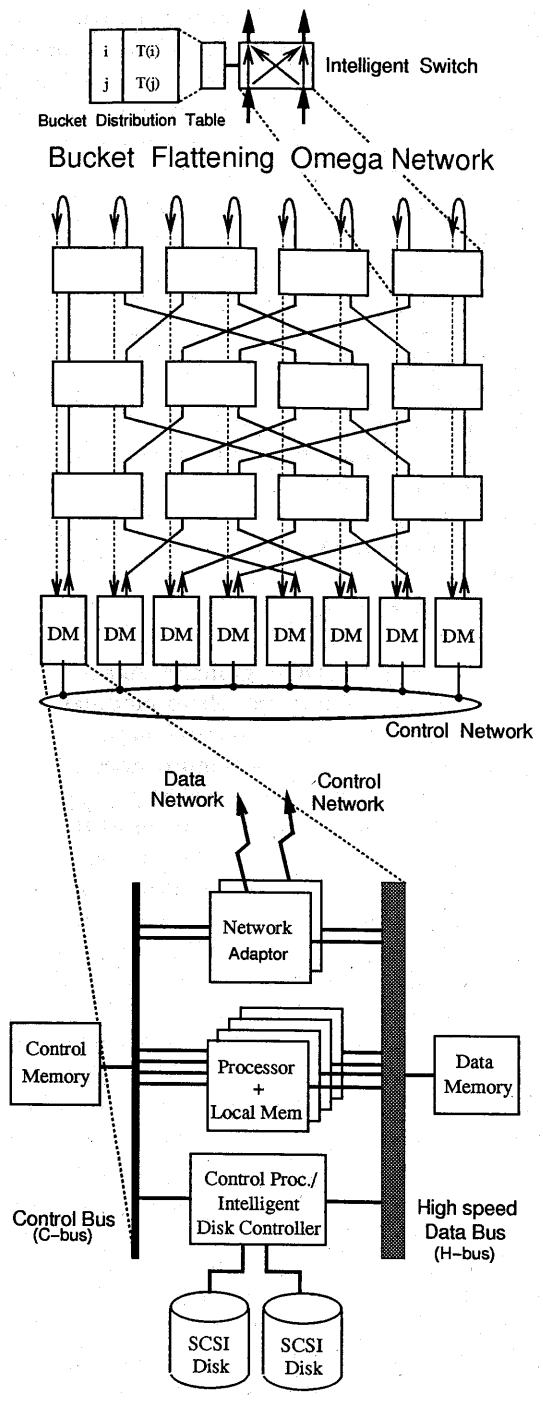


Fig. 1 Overview of super database computer I (SDC-I).

handles control information and supports communications with front-end machines. Each module has a network adaptor for each network. When invoked by the control processor, the data network adaptor asynchronously begins the data transfer from the data memory to the data network through the FIFO buffer memory on the adaptor, while processors are producing the pages of the result tuples on the data memory. In the same way, the network adaptor handles receiving tuples from the network. Although Fig. 1 shows eight modules, actually two modules were implemented and evaluated.

3.2 Functional Interconnection Network

As described above, the interconnection networks of SDC-I are composed of the data network and the control network. The data network was designed to incorporate special hardware to support flat distribution of buckets, which is the key mechanism to handle the data skew. Since the switching units in the network offer the flattening function, processing modules need not care about the data distribution.

An omega network is employed as the network topology (Fig. 1). Each switch can be set to either of two states, straight or crossed. These switches are not controlled by the centralized control manager. They set their states autonomously using only local information.

To accomplish flat distribution of the buckets, each switch keeps the value, $D(X)$ in a counter for each bucket id, X, which is the difference between the number of tuples of the X-th bucket output to the left port and output to the right port. Since the number of counters in each switch equals to the number of buckets, counters are implemented with standard RAM chips and simple ALU logic. All counters are set to zero before the query begins. When a tuple of some bucket arrives at the switch and it is given to the left output port, the counter for that bucket is incremented. If it is given to the right output port, the counter is decremented. Thus $D(X)$ represents the skew of the bucket distribution. If $D(X)>0$, more tuples have been switched to the left output port than the right port.

Let X_{left} and X_{right} denote the bucket ids of the tuples which arrived at the left and right input ports, respectively. Then $Dif=D(X_{left})-D(X_{right})$ represents the relative skew of the distribution of tuples in the bucket of X_{left} and X_{right}. In order to distribute buckets as flatly as possible over the modules, the state of the switch is set to crossed if $Dif>0$ and straight if $Dif<0$. The state of the switch can be determined arbitrarily if $Dif=0$. Figure 2 shows an example of switching behaviors. Here, the left input port receives a tuple from the m-th bucket and the right port a tuple from the n-th bucket. Since Dif is positive, the state of the switch is set to crossed.

If the tuple length is fixed and all tuples are given to the switches synchronously, this network achieves block-free transmission. Usually, SDC-I assumes a full declustering storage scheme, that is, all the relations are declustered over all the modules. The relational operators in a query are processed one by one in left-deep tree fashion or segment by segment in

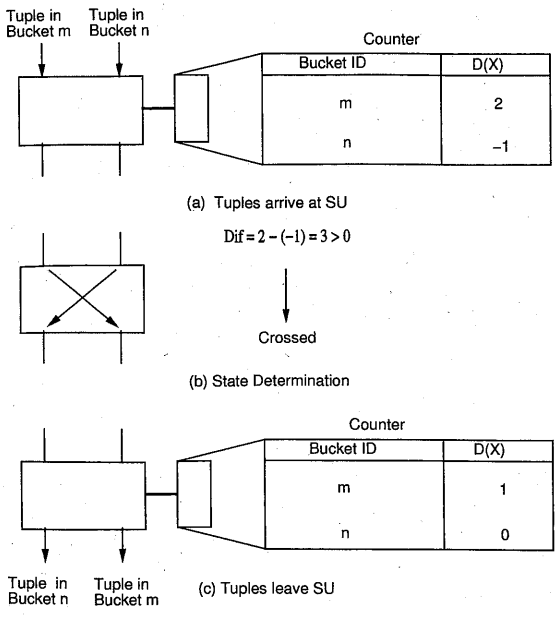


Fig. 2 State determination in a switching unit.

right-deep tree way, and block free interconnection can be fully utilized for the relations of fixed length tuples. However, in order to support variable length tuples or to support bushy tree execution parallelism, each switch has to handle asynchronous arrival of the tuples at the two input ports. The tuple which arrived earlier can determine its output port using $D(X)$. But the tuple which comes later has no choice, when the transfer of the primary tuple is still in progress. To output this tuple to the remaining port might increase the skew. In such cases, the switch blocks the transmission. Thus, asynchronous switching incurs blocking, while synchronous switching has no blocking. We introduced a threshold value to decrease the blocking ratio at the cost of increased skew. That is, when there is a possibility of blocking, each switch tests $D(X) - Thr$ instead of just $D(X)$. Thr is, in fact, a function of the current state of the switch, taking the value T if the unused output port is the left port, and the value $-T$ if it is the right one, where T is a positive constant number. A large T makes the value of the test expression increase or decrease according to whether the unused output port is the right or left one respectively, and causes blocking to be less likely to occur. Thus the value T is a relative penalty of blocking to data skew.

In SDC-II, we are planning to add two more extensions to the bucket flattening function to support more general environments. One extension is support for variable length tuples. The counter is incremented or decremented by the tuple length in bytes after the switch state was determined. As a consequence, all the modules can receive almost equal volume of tuples, rather than equal number of tuples. This avoids bucket overflow due to the skew in the data volume and eases the management of the memory space. The other is to allow the number of modules to be an arbitrary number (i.e. other than a power of 2). Usually, the number of the modules should exactly equal to a power of 2, because of the nature of the multi-stage network. However, such a restriction causes the inflexibility of system configuration. The user may want to construct the system with arbitrary number of modules. For the system with n modules where $2^{k-1} < n \leqq 2^k$, we use k stage omega network with $2^k - n$ ports disabled. We introduced another parameter representing the number of active ports, and use it as a weighting factor so that the unequal number (or volume) of tuples can be output from the right and the left ports. The details of the algorithms are beyond the scope of this paper, and will be described in the future paper.

4. System Software

The software system of SDC should be designed so that the bucket spreading hash algorithm can run as efficiently as possible. Parallel hash based relational database processing algorithms require that the tuples always flow through the modules. While the computation load is heavy for scientific applications, for database processing, data movement over the memory and the network is much more intensive than computation. Efficient data movement is the largest concern in I/O intensive parallel database processing. SDC adopts an I/O driven processing model. An abstract view of data movement in SDC-I is depicted in Fig. 3, where dark circles denote tasks composed of a filled page and white circles those with an empty page. The task are generated by the disks and are put on the task pipe over the shared memory. The processors represented by four rectangles at the center of the figure, fetch the task from the pipe, perform relational algebraic operations on it, produce result tuples and apply the hash function for the next operation if necessary, and finally release the page which was occupied by task to the free page pool. The tasks generated by the processors are sent out to the other modules over the network or written back to their own disks. The programmer writing the relational processing code does not have to be concerned with the details of flow control.

SDC Operating System (SDC-OS) which was developed as the fundamental system software for SDC-I provides only the following four primitives to the programmer,

getTask ( ): get a task from the input pipe
putFree( ): return an empty page to the free pool
getFree( ): get an empty page for result tuples from the free pool
putTask( ): put a task into the output pipe

Conventional operating systems suffer from the

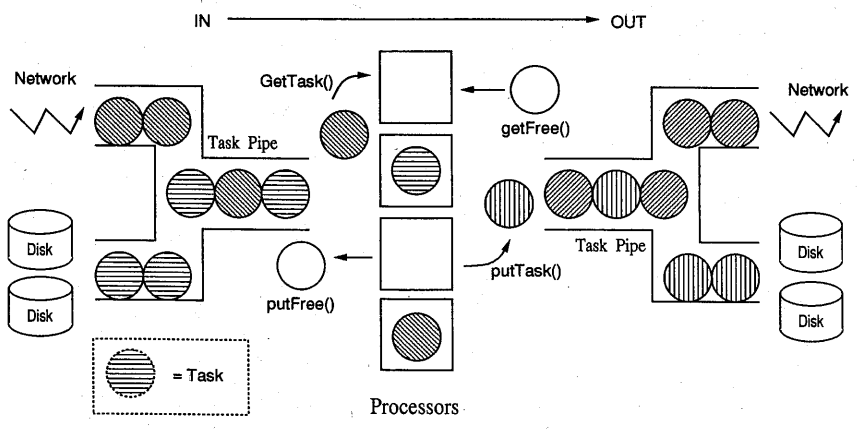


Fig. 3 SDC-OS process model.

data copy overhead between the various layers of software, for example between kernel space and user space. SDC-OS was carefully designed to reduce this overhead as much as possible. The disk controller and network adaptor incorporates task generation hardware. The tuple stream from the disk/network is transformed into the page format by hardware, which are then placed on the task pipe. During the data transfer, there is no explicit copy operation. Only the pointers to the pages are exchanged among the processors, the disk controller, and the network adaptor via the control memory.

Flow control and buffer management are also important issues for parallel database processing, which are handled by SDC-OS. There are four kinds of buffers: a read buffer for the disks and the networks, a net buffer, a write buffer, and a bucket buffer for the hash table. A high water mark is introduced on the net buffer. If the number of tuples in the net buffer exceeds the high water mark, the disks are suspended from reading the tuples, waiting for the target modules to consume the tuples transmitted by this module. The disk read operation resumes when the number of tuples in the net buffer falls below the low water mark. While the disk reads are suspended, the tuples in the write buffer are flushed out to disk. Thus the data flows in SDC are controlled so as to keep the disks idle time to a minimum. Our adaptive flow control scheme works well even under dynamically varying loads, which are confirmed by the detailed simulation [6]. Usually the disk I/O and the network I/O are treated completely independently. However as shown in Fig. 3, the tasks from both the disks and the network are equivalently processed in our application. To ease the programming, these two I/O's are unified. There is no distinction between these two, which is also one of the features of SDC-OS.

On top of the SDC-OS, SDC-DB was constructed, which is a collection of processes dedicated to each database management function. The processors of each module run data processing processes, which perform the relational algebraic operations using the four primitives described above. On the control processor, several processes such as Module Control Process, Disk Manager, Network Manager Process, Memory & DMA manager, and SCSI driver are activated. The Module Control Process is responsible for all the activities occurring in a single module. SDC-I is connected to the server machine, where the SQL compiler, the scheduler and the coordinator are invoked. The last one synchronizes the operations among the multiple modules. Barrier synchronization is necessary for phase transitions such as the switch from the i-th bucket to the $(i+1)$-th bucket, and the transition from the build phase to the probe phase.

5. Performance Monitor

It is usually difficult to understand the behavior of parallel processing systems, since so many activities run simultaneously. This stimulates the research on the performance monitoring tool and its visualization system. The majority of such tools developed so far are for scientific applications and focus on just CPU utilization. Since database processing requires large amount of I/O to the secondary storage system, I/O behavior and buffer memory utilization are also influential factors to the overall performance.

The SDC performance evaluation tool consists of performance measurement tools and performance visualization tools. There are two approaches for the performance monitor: hardware monitor and software monitor. The former can measure the system with minimum interference but its dedicated hardware incurs a high cost. The latter is easy to introduce but usually has side effects which are not negligible. Therefore we integrated these two approaches to form the SDC performance monitoring system. We developed the bus monitor as a hardware monitor and the resource monitor as a software monitor. The bus monitor is designed to measure the utilization efficiency of the H-bus and C-bus which can be found

in Fig. 1 and examine the I/O activities of the disk drives. Since all the data from the disks flows through the bus, I/O behavior can be monitored by examining the bus. The most sensitive component in SMP is the common bus, whose bandwidth determines the total number of processors in the system. Especially in database applications, data movement is the major task, which produces a heavy load on the common bus. Thus the traffic on the bus must be carefully examined. The resources which try to issue H-bus requests are four processors, the control processor, two disk controllers, and the network adaptor. C-bus is accessed by four processors and the control processor. Thus there are in all thirteen bus grant signals, and the bus monitor can select four of them to watch at a time. The active time of these signals are cumulated by the hardware counters in the bus monitor. The monitor has two kinds of precision modes: 50 nsec and 100 nsec, where the bus cycle of the H-bus is 50 nsec. For each counter, 2 MBytes RAM are prepared to store the traffic data. The most precise measurement mode can monitor the system for 54 seconds. A 216 seconds-run can be monitored with a rougher measurement.

The resource monitor embeds the monitor routine into each of the processes and measures the CPU utilization efficiency of the four processors and the control processor. The amount of memory allocated for each buffer (read buffer, write buffer, bucket buffer, and net buffer) is also monitored by the resource monitor. The data memory space is managed in the unit of page and the memory allocation tables are kept in the control memory. The control processor can take the statistics by reading the control information on the control memory, which maintains the number of pages assigned for each buffer and free pool. The visualization tools of SDC-I consists of SDC-Tacho and SDC-View. The former is the on-line monitor tool which runs on the server machine and shows the resource utilization efficiencies whose statistics are transmitted by the resource monitor on the control processor. The monitored data is visualized as the tachometer on the window. Most of the resource activities can be summerized in just one display, which helps us understand the global behavior of SDC and aids in debugging the system software. SDC-View takes the two files produced by the bus monitor and the resource monitor. After the execution it displays the overall performance data, which are mainly used for the detail analysis. Figure 4 shows the resource utilization time chart by SDC-View. For ease of understanding, the expanded Wisconsin benchmark (1 Mtuple Join_A_Sel_B) with naive parallel hash join was run on SDC-I. Two relations, R and S are initially declustered over the two modules. In Fig. 4, the top five chart shows the CPU utilization of the control processor and the four processors. The relational database processing load is equally distributed over the four processors. The next

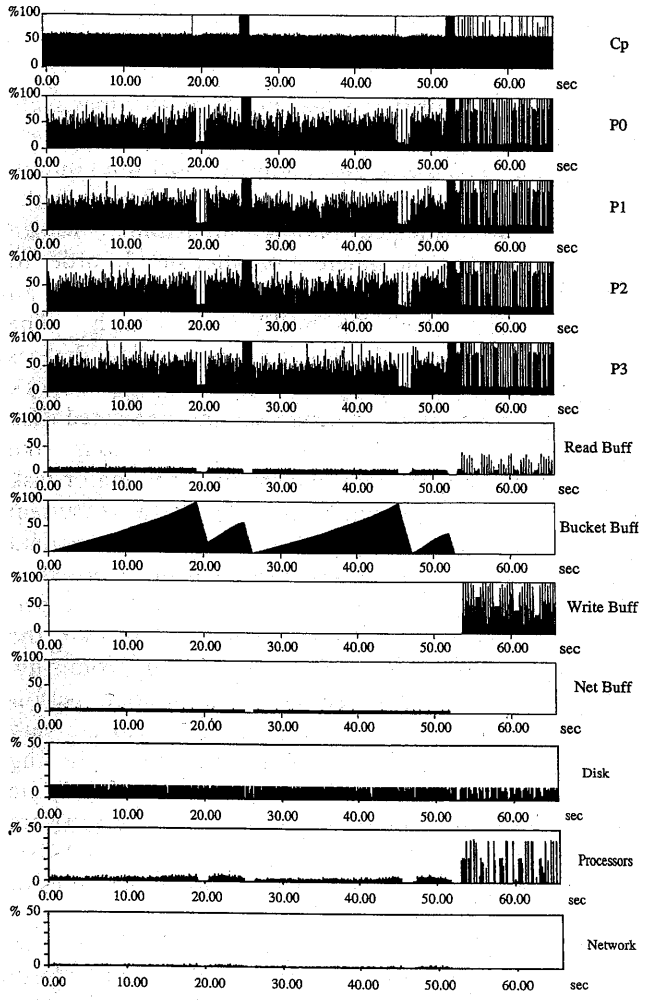


Fig. 4 Visualization by SDC-view running Join_A_Sel_B Wisconsin benchmark.

four charts, in Fig. 4, show the memory utilization of each buffer. The read buffer is almost always empty, which means that the processors are sufficiently fast to consume the pages, keeping up with the data stream from the disks. The behavior of the write buffer is like a saw. Since the write buffer has a high water mark, its contents are flushed out when the number of tuples reaches this mark. Once it goes below the low water mark, the disk read operation resumes. The last three charts shows the utilization of the H-bus. It shows that the system works so well that the disks have almost no idle time and are almost always busy. The bus is not saturated, but still has room for additional disks and processors.

As shown in Fig. 4, Join_A_Sel_B executes in 65 seconds. This performance is sufficiently high compared with the current commercial relational database systems.

6. Conclusion

This paper explores the feasibility of the massively parallel processing systems for parallel relational query processing. The query processing applications contain a high degrees of potential parallelism. Through the use of a sophisticated parallel algorithm, we can efficiently exploit the inherent parallelism. In order to examine the feasibility of parallel query processing, an experimental testbed was build from scratch. The hardware system fully utilizes commodity devices except for the special purpose interconnection network with its tuple counting mechanism and dedicated disk controller. For data intensive applications like database processing, data movement is the major task rather than the computation. The software system controls the flow of data efficiently so that the data stream from the disk be disturbed as little as possible. Although only two modules were developed, the performance evaluation results convinced us that the proposed approach is very powerful and promising. The pilot system achieved our goal of high performance. The activities in the parallel system are very difficult to understand. A performance monitoring system was also developed, which integrated both the special hardware based bus monitor and the software monitor. Visualization tool, SDC-Tacho and SDC-View were very useful to grasp the global activities of the system.

The modular architecture gives the ability to flexibly scale the system and the bucket spreading hash join algorithm allows an almost linear increase in performance. We do not say that this is the ultimate architecture for parallel query processing. Current technology increases the performance of the microprocessors and decreases its price dramatically due to mass production. In order to increase the performance/cost ratio, the design should empoly commodity elements as much as possible. However through advances in sophisticated CAD systems, the designer will be able to develop their own chips much more easily. Future parallel database servers are expected to integrate several super chips of special purpose hardware in them.

Acknowledgement

Mr. Stephen Davis polished up the quality of English significantly.

References

[1] Calino, Jr., F. and Kostamaa, P., "Exegesis of DBC/1012 and P-90," *Teradata Advanced Concepts Laboratory (TACL)*, 1992.

[2] DeWitt, D. J., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B. and Muralikrishna, M., "GAMMA—a high performance dataflow database machine," *Proc. of the 12th Int. Conf. on VLDB*, pp. 228-237, 1986.

[3] Faudemay, P. and Mhiri, M., "An associative accelerator for large databases," *IEEE MICRO*, pp. 22-34, 1991.

[4] Fushimi, S., Takeda, Y., Iwasaki, H., Komiya, F. and Nakagome, H., "A database processor greo," *Journal of Information Processing Society of Japan*, vol. 33, no. 12, pp. 1416-1423, 1992.

[5] Graefe, G., "Encapsulation of parallelism in the volcano query processing system," *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pp. 102-111, 1990.

[6] Hirano, S. and Kitsuregawa, M., "A high performance parallel I/O model and its deadlock prevention/avoidance technique on the super database computer (SDC)," *Proc. of HICCS-26, IEEE 26th Hawaii Int. Conf. on Computer Sciences*, 1993.

[7] Hollaar, L. A., "Implementation and evaluation of a parallel text searcher for very large text databases," *Proc. of HICCS-25, IEEE 25th Hawaii Int. Conf. on Computer Sciences*, vol. 1, pp. 300-307, 1992.

[8] Hong, W. and Stonebraker, M., "Optimization of parallel query execution plans in XPRS," *Proceedings of the First International Confernece on Parallel and Distributed Information Systems*, pp. 218-225, 1991.

[9] Inoue, U., Satoh, T., Hayami, H., Takeda, H., Nakamura, T. and Fukuoka, H., "A relational database processor with hardware specialized for searching and sorting," *IEEE MICRO*, vol. 11, no. 6, pp. 61-70, 1991.

[10] Kitsuregawa, M., Hirano, S., Harada, M., Nakamura, M. and Takagi, M., "The super database computer (SDC): Architecture, algorithm and preliminary evaluation," *Proc. of HICCS-25, IEEE 25th Hawaii Int. Conf. on Computer Sciences*, pp. 308-319, 1992.

[11] Kitsuregawa, M., Nakayama, M. and Takagi, M, "The effect of bucket size tuning in the dynamic hybrid GRACE hash join method," *Proc. of Int. Conf. on VLDB.*, pp. 257-266, 1989.

[12] Kitsuregawa, M., Nakano, M., Harada, L. and Takagi, M., "Functional Disk System for relational database," *Proc. of 3rd Int. Conf. on Data Engineering*, pp. 88-95, 1987.

[13] Kitsuregawa, M., Nakano, M. and Takagi, M., "Query execution for large relations on Functional Disk System," *Proc. of 5th Int. Conf. on Data Engineering*, pp. 159-167, 1989.

[14] Kitsuregawa, M. and Ogawa, Y., "Bucket Spreading Parallel Hash: A new parallel hash join method with robustness for data skew in super database computer (SDC)," *Proc. of 16th Int. Conf. on VLDB*, pp. 210-221, 1990.

[15] Kitsuregawa, M., Tanaka, H. and Moto-oka, T., "GRACE: relational algebra machine based on hash and sort—its design concepts—," *Journal of Information Processing*, vol. 6, no. 3, pp. 148-155, 1983.

[16] Kitsuregawa, M., Yang, W., Suzuki, T. and Takagi, M., "Design and implementation of high speed pipeline merge sorter with run length tuning mechanism," *5th Intenational Workshop on Database Machines*, pp. 144-157, 1987.

[17] Kojima, K., Torii, S. and Yoshizumi, S., "IDP—a main storage based vector database processor," *5th Intenational Workshop on Database Machines, also in Database Machines and Knowledge Base Machines*, Kluwer Academic Publishers, eds. M. Kitsuregawa and H. Tanaka, pp. 47-60, 1987.

[18] Kudo, T., "The Data Base Assist," *Journal of Information Processing Society of Japan*, vol. 33, no. 12, pp. 1431

-1435, 1992. (In Japanese)

[19] Lee, K. C., Hickey, T. M., Mak, V. W. and Herman, G. E., "VLSI accelerators for large database systems," *IEEE MICRO*, pp. 8-20, 1991.

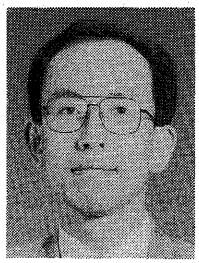
[20] Matsuda, S., Inoue, S., Shimakawa, K. and Yamada, A., "A database operation processor: DBE," *Journal of Information Processing Society of Japan*, vol. 33, no. 12, pp. 1424-1430, 1992. (In Japanese)

[21] Nakayama, M., Kitsuregawa, M. and Takagi, M., "Hash-partitioned join method using dynamic destaging strategy," *Proc. of the 14th Int. Conf. on VLDB*, pp. 468-478, 1988.

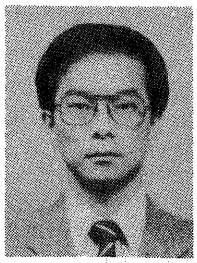
[22] Takahashi, K., Yamada, H. and Hirata, M., "A string search processor LSI," *J. Information Processing*, vol. 13, no. 2, pp. 183-189, 1990.

[23] Tsuchida, M. and Torii, S., "Integrated database processor IDP and filtering processor RDSP," *Journal of Information Processing Society of Japan*, vol. 33, no. 12, pp. 1409-1415, 1992. (In Japanese)

[24] Watson, P. and Townsend, P., "The EDS parallel relational database system, parallel database systems," *Proc. of PRISMA Workshop*, pp. 149-166, 1991.



Masaru Kitsuregawa received the B.S. in electronic engineering in 1978, the Master and Doctor of Engineering degree in information engineering from the University of Tokyo, in 1980 and 1983 respectively. In 1983 he joined the Institute of Industrial Science, the Univ. of Tokyo as a lecturer. He is currently an associate professor. His research for the last 10 years has been directed toward the design of high performance relational database system such as parallel hash join algorithm, its skew handler, intelligent secondary storage named functional disk system, highly parallel architecture for relational SQL server named SDC (super database computer), high speed hardware sort, disk array, persistent programming system and KD join etc. He is on editorial board of VLDB Journal, Int. Journal of Distributed and Parallel Databases.



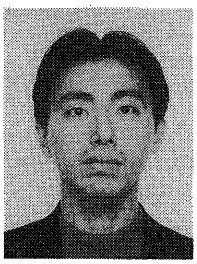
Weikang Yang received BS in engineering from the Dept. of Computer Science & Technology, Tsinghua University in Beijing in 1982, an MS in engineering Science from the University of Tokoyo in 1985, a Ph.D from the University of Tokyo in 1989. He is a member of Information Processing Society of Japan. He is working in Yozan Inc. now.

Satoshi Hirano received B.E. and M.E. degree from University of Electro-Communications, Japan in 1985 and 1987 respectively and D.E. degree from the University of Tokyo, Japan in 1992. In 1992, he joined Electrotechnical Laboratory, Japan. His current research interest includes Massively Parallel Operating System. He is a member of Information Processing Society of Japan and IEEE.

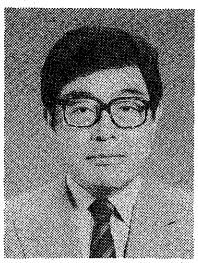
Masanobu Harada was born in Hiroshima, Japan, in 1963. Graduated National Defense Academy, in 1986 Received M.E. degree from the University of Tokyo, Japan in 1989. Currently, he is a research associate at the Dapartment of Environmental Systems Engineering, the Nagaoka University of Technology. His research interest are in parallel processing, and computer architecture.

Minoru Nakamura received B.E. and M.E. degrees from University of Electro-Communications, Japan in 1987 and 1989, respectively. Currently, he is a graduate student at Doctoral Course of Department of Engineering, the University of Tokyo, Japan. His research interest includes highly parallel computer architecture and its system software. He is a member of Information Processing Society of Japan.

Kazuhiro Suzuki was born in Hamamatsu, Japan, in 1966. He received the B.E. degree from Yokohama National University in 1990, and received the M.E. degree from the University of Tokyo in 1992. He joined Fujitsu Laboratories Ltd, in 1992. His research interests include parallel computing and GUI environment of parallel machines.



Takayuki Tamura was born in Saitama, Japan, on October 21, 1968. He received the B.E. degree in electronic engineering and the M.E. degree in information engineering from the University of Tokyo, Tokyo, Japan, in 1991 and 1993, respectively. He is now working towards the D.E. degree at the University of Tokyo. His research interests and activities are in designing interconnection networks with load balancing mechanism and its implementation on FPGA.



Mikio Takagi received the B.S. in Electrical Engineering, the M.S. and Ph.D. in Electronic Engineering from the University of Tokyo, 1960, 1962, and 1965, respectively. In 1965 he joined the Institute of Industrial Science, the University of Tokyo, as an Associate Professor. Currently, he is Professor of the Department of Electrical Engineering and Electronics and Head of the Center for Function-oriented Electronics, and Adjunct Professor of the Institute of Space and Astronautical Science. From 1971 to 1972, he was a research associate at the University of California, Santa Barbara on leave of the absence from the University of Tokyo. His research interests include digital image processing, pattern recognition, advanced architecture for multidimentional image processing, and applications of digital image processing and pattern recognition for remote sensing, medicine, and industries.